

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**

(51)Int.Cl. ⁸	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 12/00	5 3 5		G 0 6 F 12/00	5 3 5 Z
	5 4 5			5 4 5 A
15/00	3 1 0		15/00	3 1 0 U
15/16	3 7 0		15/16	3 7 0 N

審査請求 未請求 請求項の数5 F D (全 27 頁)

(21)出願番号 特願平7-211279

(22)出願日 平成7年(1995)7月28日

(31)優先権主張番号 2 8 2 6 8 3

(32)優先日 1994年7月29日

(33)優先権主張国 米国 (U S)

(71)出願人 390035493

エイ・ティ・アンド・ティ・コーポレーション

AT&T CORP.

アメリカ合衆国 10013-2412 ニューヨーク
ニューヨーク アヴェニュー オブ
ジ アメリカズ 32

(72)発明者 チュン ハー ハーマン ラオ

アメリカ合衆国, 08820 ニュージャージー
ー, エジソン, スプリングブルック ドラ
イブ 4304

(74)代理人 弁理士 三俣 弘文

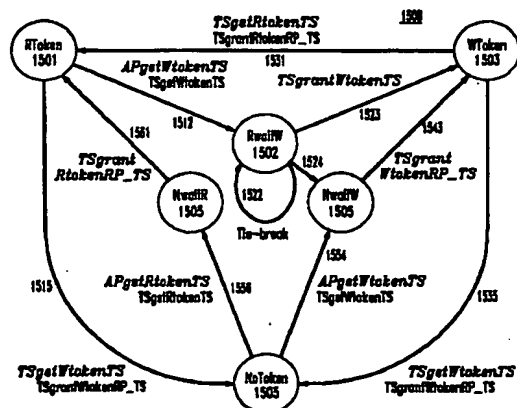
最終頁に続く

(54)【発明の名称】 分散コンピューティングシステム

(57)【要約】

【課題】 ファイルの複数のコピーを許容するがファイルシステムの意味規則はファイルの単一のコピーのものであるような分散システムを実現する。

【解決手段】 分散システムにおいて、「複製ファイル」として指定されるファイルは、分散システム内に複数のコピーが存在して、各コピーが他のコピーと完全に同等とされる。すなわち、いずれのコピーへのアクセスも、そのファイルのただ1つのローカルコピーが存在しすべてのアクセスプロセスが同一のホスト上で実行される場合のアクセスと同一の結果となる。複製ファイルのコピーに対する書き込み操作の同期は、要素システムが、複製ファイルを変更する操作を、複製ファイルのコピーに対して実行するときはいつでも、同一の操作を指定するメッセージを、その複製ファイルのコピーを有する他のすべての要素システムに送ることにより実現される。



【特許請求の範囲】

【請求項1】 分散データ処理システムの要素上で実行されるプロセスが資源に対して実行する操作を同期させる方法において、

前記プロセスにおいて、前記操作を実行するために要求されるトークンが前記要素内にあるかどうかを判断するステップと、

前記プロセスにおいて、前記トークンが前記要素内にある場合、前記トークンが利用可能になるまで待機して利用可能になると前記操作を実行し、前記トークンが前記要素内でない場合、前記要素内のトークンサーバが前記分散データ処理システム内の他の要素に前記トークンを要求するステップと、

前記プロセスにおいて、前記トークンが前記他の要素から到着したとき、前記トークンが利用可能になると前記操作を実行する操作実行ステップと、

前記トークンサーバにおいて、前記操作の完了後にはじめて前記他の要素への前記トークンに対する要求にตอบสนองするステップとからなることを特徴とする、分散データ処理システムの要素上で実行されるプロセスが資源に対して実行する操作を同期させる方法。

【請求項2】 前記操作実行ステップが、前記プロセスが、前記操作によって要求される資源に対するロックを取得するまで前記プロセスを停止し、その後で前記操作を実行するステップと、前記操作を実行した後に前記ロックを解放するステップとを有することを特徴とする請求項1の方法。

【請求項3】 前記操作が、前記他の要素において複製された、前記要素内のファイルに対する書き込み操作であることを特徴とする請求項2の方法。

【請求項4】 分散システム内に複製ファイルを生成する装置において、

前記複製ファイルの第1のコピーに対して操作を実行する第1ローカル操作実行手段と、

前記複製ファイルの第1のコピーに対して前記操作を実行する第2ローカル操作実行手段と、

各ローカル操作実行手段において、自己のファイルシステム手段で作成された複製ファイルのコピーに対する操作を他方のローカル操作実行手段に通知する操作通知手段と、

各ローカル操作実行手段において、他方のローカル操作実行手段内の操作通知手段にตอบสนองして、当該ローカル操作実行手段内の複製ファイルのコピーに対して、通知された操作を実行する通知操作実行手段と、

各ローカル操作実行手段において、複製ファイルのコピーに対する操作がどのコピーに対して実行されるかにかかわらず同じ結果を有するように複製ファイルのコピーに対する操作を同期させる手段とからなることを特徴とする、分散システム内に複製ファイルを生成する装置。

【請求項5】 複数の要素コンピューティングシステム

を有する分散コンピューティングシステムにおいて、要素コンピューティングシステム内に配置され、相異なる要素コンピューティングシステム上にコピーを有する複製ファイルを、コピーに対して実行される操作がどのコピーに対して実行されるかにかかわらず同じ結果を有するように維持する分散手段からなることを特徴とする分散コンピューティングシステム。

【発明の詳細な説明】

【0001】

10 【発明の属する技術分野】本発明は、コンピュータシステムに関し、特に、緩く接続された分散システムにおける動作の同期に関する。

【0002】

【従来の技術】元来、コンピュータシステムは、単一のプロセッサと、ファイルを記憶するディスクドライブのような大容量記憶装置とから構成された。プロセッサのユーザはそれぞれプロセッサに接続された端末を有し、そのプロセッサを使用してファイルにアクセスすることができた。このようなシステムでは、すべてのアクセスはシステムの単一のプロセッサを通じてのものであり、システムには与えられたファイルの単一のコピーのみが存在した。プロセッサ、メモリ、および大容量記憶装置（例えばディスクドライブ）の価格が低下するにつれて、このようなシングルプロセッサシステムは分散システムによって置き換えられている。分散システムには、ネットワークへと接続されたいくつものローカルなプロセッサおよび記憶装置がある。このような分散システムの簡単な例としては、各ユーザがパーソナルコンピュータまたはワークステーションを有し、ワークステーションにファイルを提供するファイルサーバがあり、ワークステーションがローカルエリアネットワークによって相互におよびファイルサーバに接続されたものがある。もちろん、このような分散システムはそれ自体、より大きな分散システムの要素となることもあり、その場合には、他のシングルプロセッサシステムおよび分散システムが、ローカルエリアネットワークまたは電話システムのような広域ネットワークによって接続される。

【0003】このような分散システムの利点には、ロバストネス（頑強さ）および速度が改善されることがある。ロバストネスの改善は、システムを構成する要素の数から生じる。例えば、1つのワークステーションが故障しても、ネットワーク上の他のワークステーションが役に立たなくなることはない。さらに、各ワークステーションは固有のファイルシステムを有し、そのため、システムにはファイルの複数のコピーが存在することがある。ファイルの1つのコピーが使用不能になった場合、他のワークステーションからコピーを使用可能である。速度の改善は、ユーザがローカルな処理能力および他人と共有していないローカルな記憶装置を有するというところから生じる。

【0004】ファイルのローカルなコピーは分散システムのロバストネスおよび速度を改善するが、書き込み可能なファイルのローカルなコピーは1つの主要な欠点を有する。それらのコピーの相互の一貫性を保つことが困難であるという欠点である。理想的には、分散システムにおけるファイル操作の意味規則（セマンティクス）は、プロセスのグループがプロセッサを共有するようなシステムにおけるものと同一であろう。例えば、UNIXオペレーティングシステムでは（UNIXはX/OPE N財団の商標である）、プロセスがデータをファイルに書き込むとき、他のプロセスは、最初のプロセスによる書き込みが完了するまで、そのファイルからデータを読み出すことも、そのファイルにデータを書き込むこともできない。これは、読み出しプロセスがファイルをオープンしたのが、書き込みプロセスがシステムコールを実行する前か後かにかかわらず成り立つ。

【0005】利用可能な分散システムはローカルコピーの問題を「キャッシュ」によって処理している。サーバにファイルの単一の主コピー（マスタコピー）があり、そのファイルを使用しているワークステーションはそのファイルの一部または全部を含むキャッシュを有する。キャッシュされたコピーにおける変化はマスタコピーには即時には反映されず、その逆もそうであるため、キャッシュされたコピーは相互にあるいは主コピーと矛盾することがある。その結果、あるワークステーションによるファイルの読み出しが他のワークステーションによる書き込みの後であっても、その読み出しは、キャッシュされたコピーを使用しているため、書き込みの結果を見ないことがある。

【0006】キャッシュを使用するシステムのファイル操作の意味規則は一貫性の欠如を反映している。このようなシステムの一例は、アール・サンドバーグ(R. Sandberg)他、「Sun Network File Systemの設計と実装(Design and Implementation of the Sun Network File System)」、Proceedings of Summer Usenix (1985年6月)第119～130ページ、に記載されたSun Network File System (NFS) である。NFSでは、与えられたファイルはファイルサーバに存在する。ワークステーションはその与えられたファイルのコピーを含むキャッシュを有することが可能である。キャッシュ内のコピーは、そのコピーがキャッシュにロードされた後3秒間はファイルサーバ内のコピーと同一であると仮定される。キャッシュされたディレクトリデータは、フェッチされた後30秒間は有効であると仮定される。

【0007】もう1つの例は、エム・カザー(M. Kazar)、「アンドリュウファイルシステムにおける同期とキャッシュの問題(Synchronization and caching issues in the andrew file system)」、Proceedings of Winter Usenix (1988年)に記載されたアンドリュウファイルシステム

は、ファイルシステムコールが完了した後、そのファイルシステムコールの結果は、2つの重要な例外を除いて、ネットワークのどこでも即時に見えることを保証する。第1の例外は、書き込みシステムコールの場合、ファイルに書き込まれた新しいデータは、そのファイルに書き込みをしているプロセスがファイルを閉じるまでは、主コピーには実際には書き込まれないことである。第2の例外は、ファイルのキャッシュされたコピーは、プロセスがそのファイルをオープンするときに主コピーとの一貫性が検査されるだけである。従って、2つのプロセスがファイルを共有するとき、あるワークステーションで実行中の第1のプロセスは、他のワークステーションで実行中の第2のプロセスによって書き込まれたデータを、第2のプロセスがそのファイルをクローズした後に第1のプロセスがそれをオープンするまでは、見ないことになる。

【0008】

【発明が解決しようとする課題】ファイルの複数のコピーを許容するがファイルシステムの意味規則はファイルの単一のコピーのものであるような分散システムが必要とされている。本発明の目的は、このような分散システムを実現することである。

【0009】

【課題を解決するための手段】本発明の分散システムでは、いくつかのファイルが「複製ファイル」として指定される。複製ファイルは、分散システム内に複数のコピーが存在して、各コピーが他のコピーと完全に同等であるようなファイルである。ファイルが完全に同等であるとは、いずれのコピーへのアクセスも、そのファイルのただ1つのローカルコピーのみが存在しすべてのアクセスプロセスが同一のホスト上で実行される場合のアクセスと同一の結果となることである。こうして、本発明の分散システムは、プロセスが同一のホストで実行されるようなシステム上の単一のファイルへのアクセスのファイルシステム意味規則と、ファイルのローカルコピーとの利点を合わせ持っている。

【0010】本発明のもう1つの特徴は、複製ファイルのコピーに対する書き込み操作を同期する技術にある。分散システムホストの要素システムが、複製ファイルを変更する操作を、複製ファイルのコピーに対して実行するときはいつでも、同一の操作を指定するメッセージが、その複製ファイルのコピーを有する他のすべての要素システムに送られる。すべてのメッセージは同一のチャネルを通じて移動するため、操作の順序はすべてのコピーに対して同一である。書き込み操作は、基本書き込み操作であることも可能であり、あるいは、複製ファイルに含まれるデータに関する操作の指定であることも可能である。

【0011】もう1つの特徴は、複製ファイルに対する順次読み出し操作が書き込み操作と同期していることで

あり、その結果、シングルプロセッサシステムにおける通常のファイルに対して実行される読み出し操作と同一の意味規則を有することになる。

【0012】順次読み出し操作と書き込み操作の同期は、各複製ファイルに対する読み出しトークンおよび書き込みトークンと、分散システムの各要素内のトークンサーバとを使用する分散同期システムによって実現される。複製ファイルに対して書き込み操作を実行するシステムの要素は書き込みトークンを有していなければならず、順次読み出し操作を実行する要素は、読み出しトークンまたは書き込みトークンを有していなければならない。要素がトークンを取得するには、その要素のトークンサーバが他の要素のトークンサーバにそのトークンを要求する。トークンサーバは、未完了の書き込み操作を指定するすべてのメッセージを他の要素から受信した後、のみそのトークンを受信する。その結果、書き込み操作はすべてのローカルコピーに対して同じ順序で実行され、順次読み出し操作はすべてのコピーに対して同じ結果を有する。

【0013】本発明の同期システムのもう1つの特徴は、複製ファイルの各ローカルコピーに対する7個のロックを使用して実装されることである。これらのロックは、標準的なオペレーティングシステムの共有ロックおよび排他ロックを使用して順に実装される。

【0014】本発明のさらにもう1つの特徴は、複製ファイルへのアクセスのトランザクショナル同期をサポートするために使用することができることである。

【0015】本発明の利点は、複製ファイルを保守するために必要な操作が、分散システムのユーザレベルで実装可能であることにある。その結果、本発明は、特殊なハードウェアや特殊なオペレーティングシステムを必要としない。好ましい実施の形態は、ユーザレベルのバックアップファイルシステムの変更として実装される。

【0016】

【発明の実施の形態】

【ライブラリを使用したインタフェースの変更：図2】既に説明したように、コンピュータシステムは層化されている。各層は隣の上位層へのインタフェースを提供する。上位層は、下位層のインタフェースによって要求されるように下位層が実行する操作を指定する。上位層が下位層によって要求されるインタフェースに従っていない場合、上位層と下位層の間にアダプタ層を追加しなければならない。アダプタ層の目的は、上位層によって期待されるインタフェースに従ってなされる操作指定を、下位層のインタフェースによって要求される操作指定に変換することである。この技術を使用して、例えば、MSDOSオペレーティングシステムを実行しているPCが、ユーザには、UNIXオペレーティングシステムを実行しているコンピュータであるかのように見えるようにすることが可能である。

【0017】アダプタ層が多くのアプリケーションプログラムによって要求されるとき、これはライブラリルーチンのセットとして実装されることが多い。その名前からわかるように、ライブラリルーチンは、コンピュータシステムのサブシステムのメーカが、そのコンピュータシステムのユーザに対して、アプリケーションプログラムとともに使用するよう提供されるルーチンのことである。図2に、どのようにしてライブラリルーチンがアダプタ層をなすように使用されるかを示す。ユーザプログラム201は、次の層（この場合には、システムルーチンのセット）へのインタフェース206を有する。しかし、ユーザプログラム201が使用されるコンピュータシステムのシステムルーチンは、インタフェース213を有する。インタフェース206とインタフェース213の相違は、図2では、インタフェースを表す線の形の相違によって表されている。アダプタ層はライブラリルーチン207からなり、ユーザプログラム201によって要求される隣の上位層に対するインタフェース206と、システムルーチン205によって要求される隣の下位層に対するインタフェース213とを有する。インタフェースは実際にはファンクション（関数）呼出しからなり、ライブラリルーチン207内のルーチンは、インタフェース206によって要求されるファンクション呼出し203に回答して、インタフェース213によって要求されるファンクション呼出しを生成し、ファンクション呼出し203によって指定される操作を実行することによって動作する。システムルーチン215は、終了すると、その実行の結果を矢印211で示されるようにライブラリルーチン207に返し、続いてライブラリルーチン211はその結果を復帰205によって示されるようにユーザプログラム201に返す。

【0018】【動的リンクライブラリルーチンを使用したインタフェースの再定義】インタフェースを再定義するためのライブラリルーチンの有用性は、従来のシステムでは、ユーザプログラム201に対する実行可能コードが生成されるときにユーザプログラム201にリンクされなければならないということによって制限されていた。この場合のリンクとは、ユーザプログラム201におけるライブラリルーチンの呼出しが、ライブラリルーチン207のコピー内のライブラリルーチンの位置に関係づけられるプロセスをいう。リンクは実行可能コードが生成されるときに行われなければならないため、実行可能コードのコピーしか有しないユーザは、あるライブラリルーチン207の他のライブラリルーチン207のセットと置換することは不可能であった。

【0019】現在ではコンピュータシステムは発展してきており、ライブラリルーチンはユーザプログラムに動的にリンクすることが可能である。このようなコンピュータシステムでは、リンクは、ユーザプログラムを実行するプロセスが実行前にコンピュータシステムのメモリ

にロードされるときに行われる。動的リンクにより、ユーザプログラムのオブジェクトコードを変更せずに、あるライブラリルーチンのセットを他のセットと置換することが可能であり、それによって、ユーザプログラムが動作するシステムの挙動を変えることが可能である。動的リンクについての説明は、「共有ライブラリ(Shared Libraries)」、Sun Microsystems, Inc.、米国カリフォルニア州マウンテン・ビュー(1988年5月)、に記載されている。

【0020】図3に、どのようにして動的リンクを使用してシステムの挙動を変えるかを示す。システム1(301)において、ユーザプロセス306はアプリケーションプログラム309を実行しており、これに、オペレーティングシステムライブラリ1(315)が動的にバインドされている。オペレーティングシステムライブラリ1(315)は、コール311および復帰313によって示されるアプリケーションプログラム309へのインタフェースを提供し、カーネルサーバ305へのコール317およびカーネルサーバ305からの復帰319を使用してコール311によって指定される操作を実行する。システム2では、ユーザプロセス306は同じアプリケーションプログラム309を実行し同じカーネルサーバ305を使用しているが、こちらでは、オペレーティングシステムライブラリ2(321)によってオペレーティングシステムライブラリ1(315)が置換されている。オペレーティングシステムライブラリ2(321)は、オペレーティングシステムライブラリ1(315)が行うすべてのことを行う。すなわち、システム301を、システム301のように挙動するが副次的効果323も生成するシステム303へと変換するのに必要なことは、オペレーティングシステムライブラリ2(321)をオペレーティングシステムライブラリ1(315)の代わりにユーザプログラム309に動的にリンクすることだけである。

【0021】[動的リンクライブラリを使用したユーザレベル名前空間の作成]さらに、図4に、どのようにして動的リンクオペレーティングシステムライブラリ403を使用してユーザレベル名前空間405を作成し、どのようにしてユーザレベル名前空間405を使用して副次的効果323を制御するかを示す。ファンクション、ファイルおよびデバイスのようなコンピュータシステムにおけるエンティティはプログラムにおいて名前によって呼ばれ、プログラムで使用される名前をその名前によって表されるエンティティに関係づけることはコンピュータシステムの名前空間の機能である。従来のコンピュータシステムでは、ユーザプログラムによって使用される名前空間はオペレーティングシステムによって作成され保守されている。システム401では、オペレーティングシステムライブラリ403がユーザプロセス409に対する1つ以上のユーザレベル名前空間405を作成

し保守する。ユーザレベル名前空間405がライブラリルーチン403によって使用されることを可能にする1つの方法は、カーネルサーバ305によってユーザプログラム309に提供されるファイルシステムとは挙動、構造、またはその両方において異なるユーザレベルの論理ファイルシステムを作成することである。その後、この論理ファイルシステムを使用して、副次的効果323を制御する。例えば、システム401がバックアップファイルシステムである場合、副次的効果323は、バックアップファイルシステムを生成するために要求されるものであり、ユーザレベル名前空間405は、カーネルサーバ305によって提供されるファイルシステムにおけるどのファイルがバックアップファイルシステムにバックアップされるべきかを指定することが可能である。図4から明らかなように、ユーザレベル名前空間405はユーザプロセス409の環境の一部である。

【0022】[ユーザレベルバックアップファイルシステムの概観: 図5~図6]上記の動的リンクライブラリおよびユーザレベル名前空間を使用して、アプリケーションプログラムを実行しているアプリケーションプロセスによって変更されたファイルのうち選択したもののみを自動的にバックアップするユーザレベルバックアップファイルシステムを形成することが可能である。図5に、そのようなユーザレベルバックアップファイルシステム501を示す。システム501は、2つのコンピュータシステムによって実装される。主システム511では、アプリケーションプロセス503が実行され、バックアップシステム513では、アプリケーションプロセス503によって変更されたファイルのバックアップコピーが保守される。主システム511およびバックアップシステム513は通信媒体によって接続され、これによって、主システム511で実行されているプロセスからのメッセージをバックアップシステム513で実行されているプロセスへ送ることができる。

【0023】主システム511上のシステム501の要素は、アプリケーションプロセス503およびカーネルサーバ305(a)である。カーネルサーバ305

(a)は主システム511にファイルシステムを提供する。図5において、ファイルシステムは主システム511に対してローカルなディスク307(a)によって表されているが、これは他のシステム上に位置するリモートファイルシステムでも全くかまわない。いずれの場合にも、カーネルサーバ305(a)は、アプリケーションプロセス503からのコール317に回答して、提供するファイルシステムに対するファイル操作を実行し、結果319をプロセス503に返し、自分自身必要な操作をディスク307(a)に対して実行する。アプリケーションプロセス503は、動的リンク可能ライブラリを使用して、カーネルサーバ305(a)とともにファイル操作を実行する。主システム511では、このライ

ブラリは、lib. 3d (507) と呼ばれる新たなライブラリによって置換されている。ライブラリ507は、いくつかのファイルを変更するファイル操作を指定するコール311に回答して、カーネルサーバ305への適当なコール317を提供するだけでなく、バックアップメッセージ512をバックアップシステム513に送る。変更の結果バックアップメッセージ512を送ることになるファイルはフロントエンド複製ツリー (FRT) 505で指定される。複製ツリー505は、矢印506で示されるように、lib. 3d (507) 内のルーチンによって保守され使用される。このようにして、複製ツリー505は、変更の結果システム513上のバックアップファイルを変更することになるファイルからなるユーザレベル論理ファイルシステムを定義する。

【0024】バックアップシステム513上のシステム501の要素は、バックエンドサーバ515、ユーザレベルプロセス、ならびにカーネルサーバ305 (b) およびディスク307 (b) であり、バックアップシステム513のための標準的なファイルシステムサーバおよびディスクドライブである。カーネルサーバ305

(b) はバックエンドサーバ515にファイルシステムを提供する。図5では、ファイルシステムのためのデータはローカルディスク307 (b) 上に記憶されている。しかし、これはリモートシステムに記憶することも可能である。バックエンドサーバ515は、カーネルサーバ305 (b) へのコール317によってファイル操作を実行し、そのコールの結果をサーバ305 (b) から受け取る。バックエンドサーバ515はバックエンドマップ517を保守する。バックエンドマップ517は、フロントエンド複製ツリー505によって指定されるファイルを、バックアップとして使用されるバックアップシステム513のファイルシステム内のファイル上にマップする。カーネルサーバ305 (a) によって生成されるファイルシステムとカーネルサーバ305

(b) によって生成されるファイルシステムが同一の名前空間を有するような実施の形態では、バックエンドマップ517は不要となる。

【0025】どのようにしてシステム501が動作するかは図6から明らかとなる。図6には、ファイルを変更するライブラリ507内のルーチン601の形式の一般的概略が示されている。ルーチン名603およびこのルーチンがとる引数605は、ライブラリ507によって置換されたライブラリ内のファイル操作を実行するために使用される関数の名前および引数と同一である。その結果、アプリケーションプログラム509におけるこのルーチンの呼出しはルーチン601を呼び出す。必要な準備を実行した後、ルーチン601はカーネルサーバ305 (a) に、ルーチン601によって置換されたルーチンと同じファイル操作を実行させる。この操作が成功した場合、ルーチン613は、変更されたファイルの名

前とともに関数613を呼び出し、変更されたファイルがバックアップされるべきであるということを確認する。フロントエンド複製ツリー505が示しているかどうかを判定する。フロントエンド複製ツリーがそのように示している場合、関数615は引数617によりメッセージ512をバックアップシステム513へ送る。メッセージ512は、バックアップシステム513が、サーバ305

(a) によって提供されるファイルシステム上でちょうど実行されたのと全く同じ操作をバックアップファイルシステムに対して実行するよう要求する。このメッセージを送った後、ルーチン601は復帰する。これは、ファイルがフロントエンド複製ツリー505内になかった場合、または、関数607によって指定される操作が成功しなかった場合も同様である。図6で611とラベルされているコードのセクションは、副次的効果（この場合はメッセージ512）を指定する。ここで注意すべきルーチン601の特徴は、メッセージ512がバックアップシステム513に送られるのはファイル操作が主システム511で成功した場合のみであるということである。これは、不成功の操作はバックアップする必要がないためである。

【0026】システム501には一般的に2つのクラスのファイル操作がある。フロントエンド複製ツリー505およびバックエンドマップ517によって実装されたユーザレベル名前空間405を変更するものとそうでないものである。第2のクラスの操作の一例は、フロントエンド複製ツリー505に指定されたファイルへの書き込みである。lib. 3d (507) 内の書き込みファンクションは、lib. 3dによって置換されたライブラリ内の書き込みファンクションと同じインタフェースを有する。好ましい実施の形態では、これは、引数として、ファイルを指定するためにカーネルサーバ305

(a) によって使用される整数のファイルディスクリプタと、書き込むデータを含むバッファへのポインタと、書き込むデータのサイズを示す整数とをとる。lib. 3d内の書き込みファンクションは、カーネルサーバ305 (a) が、ファイルディスクリプタによって指定されるファイルに対してシステム書き込みファンクションを実行することを要求し、その操作が成功した場合、このファンクションは、そのファイルディスクリプタによって指定されるファイルがフロントエンド複製ツリー505内に存在するかどうかを検査する。存在する場合、ファンクションはバックアップシステム513内のバックエンドサーバ515へ書き込みメッセージ512を送り復帰する。このメッセージは、カーネルサーバ305 (a) によってちょうど書き込まれたファイルを指定し、カーネルサーバ305 (a) によって提供されるファイルシステム内のシステム書き込み操作によってちょうど実行された書き込み操作を全く同様にバックアップファイルシステムにおいて実行するのに必要な情報を含

む。バックエンドサーバ515は、このメッセージを受け取ると、バックエンドマップ517を使用して、カーネルサーバ305(b)がバックアップファイルに対して使用するファイルディスクリプタを判定してから、カーネルサーバ305(b)によって提供されるシステム書き込みファンクションを使用して、このメッセージによって提供されるデータおよび位置の情報をを用いてバックアップファイルに対して書き込み操作を実行する。

【0027】ユーザレベル名前空間405を変更する操作の簡単な場合はファイル削除である。lib. 3dによって提供される削除ファンクションは、まずカーネルサーバ305(a)にファイルを削除するよう要求する。この削除が終了すると、削除ファンクションは、削除されたファイルに関する情報をフロントエンド複製ツリー505から削除することが必要かどうかを検査する。それが必要な場合、ファンクションはその情報を削除する。次に、ファンクションは、削除に必要なメッセージをバックエンドサーバ515へ送り復帰する。バックエンドサーバ515は、このメッセージを受け取ると、バックエンドマップ517内でそのファイルを見つけ、カーネルサーバ305(b)にそのファイルを削除するよう要求するとともに、この削除によって要求される操作をバックエンドマップ517に対して実行する。

【0028】より複雑な例は名前変更である。カーネルサーバ305(a)によって提供されるファイルシステム内のファイルの名前変更がユーザレベル名前空間405において引き起こす結果には3つの可能性がある。

【0029】1. そのファイルの古い名前がユーザレベル名前空間405の一部であり、新しい名前もまたユーザレベル名前空間405の一部である場合、そのファイルはユーザレベル名前空間405内にとどまる。

2. そのファイルの古い名前はユーザレベル名前空間405の一部でないが、新しい名前はユーザレベル名前空間405の一部である場合、そのファイルはユーザレベル名前空間405に追加される。

3. そのファイルの古い名前はユーザレベル名前空間405の一部であるが、新しい名前はユーザレベル名前空間405の一部でない場合、そのファイルはユーザレベル名前空間405から削除される。

【0030】第1の場合、lib. 3dの名前変更ファンクションは、カーネルサーバ305(a)に、そのファイルシステムにおける名前変更を行うよう要求する。次に、このファンクションは、名前変更されたファイルがユーザレベル名前空間405内にあるかどうかを検査し、ユーザレベル名前空間405内にある場合、名前変更ファンクションは、その変更を反映するようにフロントエンド複製ツリー505を変更し、バックエンドサーバ515における名前変更を要求するメッセージをバックエンドサーバ515へ送り、復帰する。このメッセージはもちろん、旧パス名および新パス名を含む。バック

エンドサーバ515は、このメッセージを受信すると、カーネルサーバ305(b)に名前変更を要求する。

【0031】第2の場合、名前変更ファンクションは、サーバ305(a)に名前変更を要求し、前のように、名前変更されたファイルがユーザレベル名前空間405内にあるかどうかを検査するが、今度は、ファンクションは、名前変更されたファイルをフロントエンド複製ツリー505から削除し、メッセージをバックエンドサーバ515へ送り、復帰する。バックエンドサーバ515へのメッセージは、そのファイルに対する削除メッセージである。このメッセージに回答して、バックエンドサーバ515はカーネルサーバ305(b)にバックアップファイルを削除させる。

【0032】第3の場合も、前のように、名前変更ファンクションは名前変更を要求するが、今度は、2つのメッセージを送らなければならない。第1のメッセージは、ユーザレベル名前空間405へ移動されたファイルの名前を有するファイルをバックアップシステム513内に作成することを要求する。バックエンドサーバ515はこのメッセージに回答してカーネルサーバ305

(b)がそのファイルを作成することを要求し、バックエンドマップ517内にそのファイルのエントリを作成する。その後、名前変更ファンクションはユーザレベル名前空間405に移動されたファイルの現在の内容とともに書き込みメッセージを送る。バックエンドサーバ515はこの書き込みメッセージに回答して、カーネルサーバ305(b)によって、バックアップシステム513内のバックアップファイルにその内容を書き込む。

【0033】以上のことからわかるように、主システム511内のカーネルサーバ305(a)によって実行される単一の操作は、バックエンドサーバ505がカーネルサーバ305(b)に一連の操作を実行させることを要求する。さらに理解されるように、lib. 3d(507)内のファンクションによって実行される操作の最後には、バックエンドマップ517およびフロントエンド複製ツリー505は常に同じ状態になる。

【0034】[好ましい実施の形態の実装: 図7~図11] 図7に、ユーザレベルバックアップファイルシステムの好ましい実施の形態701の詳細ブロック図を示す。この好ましい実施の形態は、一方のプロセッサがUNIXオペレーティングシステムのSunOS4. 1バージョンを実行しており他方のプロセッサがUNIXオペレーティングシステムのMIPS4. 5バージョンを実行しているシステムにおいて実装された。システム701には要素の2つのグループがある。一方のグループの要素はバックアップファイル操作を実行し、他方のグループの要素はシステム701をフォールトトレラントにする。以下の説明では、まず、バックアップファイル操作を実行する要素について説明し、その後で、フォールトトレランスを提供する要素について説明する。

【0035】主システム511から始めると、アプリケーションプロセス503は、アプリケーションプログラム509、動的リンク可能ライブラリlib. 3d (507)、およびフロントエンド複製ツリー505を有する。ライブラリ507のファンクションはファイル操作の副次的効果としてバックアップファイル操作を実行する。システム501において、ファイル操作は、カーネルサーバ305 (a) によって実行される。ライブラリ507内のファンクションによって生成されるメッセージは、パイプ710によってバックアップシステム513へ運ばれる。パイプ710は、パイププロセス711によってアプリケーションプロセス503に提供され、パイププロセス711自体、パイプ709によってアプリケーションプロセス503と通信する。以下でさらに詳細に説明するように、パイププロセス711は、バックアップシステム513上にバックアップを作成するすべてのアプリケーションプロセス503によって使用される単一のパイプ710を提供する。

【0036】次に、好ましい実施の形態におけるバックアップシステム513において、バックエンドサーバ515は2つのプロセス、すなわち、バックエンドログプロセス (BLP) 716およびシステムコールエンジン (SYSCALL ENG) 715に分かれる。いずれもカーネルサーバ305 (b) を使用してファイル操作を実行する。バックアップファイルに加えて、カーネルサーバ305 (b) によって保守されるファイルシステムはログファイル703 (b) を含む

【0037】動作は以下の通りである。アプリケーションプロセス503は、初期化されると、パイプ710を指定するファイル識別子をパイププロセス711から取得する。アプリケーションプログラム509の実行の結果、ファイル操作が実行されると、lib. 3d (507) 内のその操作に対するファンクションは、カーネルサーバ305 (a) によって提供されるファイルシステムに対して、カーネルサーバ305 (a) にそのファンクションを実行させ、さらに、パイプ710を通じてメッセージをバックアップシステム513に送る。このメッセージは、バックアップシステム513に到着すると、バックアップログプロセス716によって受け取られる。バックアップログプロセス716は、カーネルサーバ305 (b) によって提供されるファイルシステム内のログファイル703 (b) 内にそのメッセージをログする。ログファイル703 (b) がメッセージを有するときにはいつでも、そのメッセージは、到着順に、システムコールエンジンプロセス715によって読み出される。好ましい実施の形態では、バックエンドマップ517はシステムコールエンジンプロセス715に属する。システムコールエンジンプロセス715は、メッセージを読み出すと、カーネルサーバ305 (b) に、そのメッセージによって要求されるファイル操作を実行さ

せ、システムコールエンジンプロセス715自信は、そのメッセージによって要求されるようにバックエンドマップ517を保守する。

【0038】[システム701のフォールトトレラント動作] システムのフォールトトレラント動作には、故障が検出され、検出された故障に応じてシステムが動作を継続することができるようになっていことが要求される。好ましい実施の形態では、故障の検出およびその故障への応答は、WatchDという、分散システムをフォールトトレラントにするためのユーザレベルのシステムによって扱われる。WatchDについての詳細は、ワイ、フアン(Y. Huang)、シー、キンタラ(C. Kintal a)、*「ソフトウェア実装フォールトトレラント：技術と経験(Software Implemented Fault Tolerance: Technologies and Experiences)」*、第23回フォールトトレラントコンピューティングに関する国際会議(23rd International Conference on Fault Tolerant Computing)、フランス国ツールーズ、1993年6月22～24日、に記載され、また、米国特許出願第07/954, 549号(発明者：ワイ、フアン(Y. Huang)、出願日：1992年9月30日)の主題ともなっている。本発明の説明のためには、WatchDシステムが、libftというライブラリと、分散システムの各ノード上の1つのモニタプロセスとを含むことを理解していればよい。libftは、WatchDにプロセスを登録する操作、自動バックアップ用にメモリの領域を指定する操作、および、そのメモリ領域に対してチェックポイント操作を実行する操作などを実行するルーチンを含む。モニタプロセスは、WatchDに登録されたユーザプロセスをモニタするとともに、相互をモニタする。モニタは、登録されているプロセスが故障したと判定すると、そのプロセスを再起動する。プロセスは、libftファンクションによって再起動されたときに何が起きたかを判定することが可能である。分散システムの1つのノード上のユーザプロセスをモニタする間、モニタは、重要データ(これもまたlibftファンクションを使用して定義される)のコピーを分散システムの他のノードへ移動することが可能である。そのモニタのノードが故障すると、他のノード上のモニタがその故障を検出し、重要データの現在のコピーを使用して当該他のノード上でユーザプロセスを再起動する。故障したノードが復旧すると、そのノードのモニタは、他のノードからの重要情報を使用してユーザプロセスを再起動し、ユーザプロセスが再起動されたことを示すメッセージを送る。他のノードのモニタは、そのメッセージを受け取ると、当該他のノードで実行されているユーザプロセスを終了する。一般に、WatchDモニタはリング構成で配置され、各モニタはリングにおける隣のモニタをモニタする。リング内のノードの数およびユーザプロセスの重要データのコピーを受け取るモニタの数は、WatchDに登録さ

れたユーザプロセスを再起動することができなくなる前に分散システムのいくつかのノードが故障しなければならぬかを決定する。

【0039】好ましい実施の形態では、主システム511およびバックアップシステム513はそれぞれWatchDモニタを有する。これらのモニタとシステム701の要素の間の関係は、破線矢印721で示されている。主システム511のモニタはモニタ717である。破線矢印721で示されるように、モニタ717は、パイププロセス711、フロントエンドログプロセス705、およびシステム513内のモニタ719を監視する。モニタ719は、モニタ717、システムコールエンジンプロセス715、およびバックエンドログプロセス716を監視する。

【0040】図7に示されるように、システム701は、フロントエンドログプロセス705、パイププロセス711、システムコールエンジン715、バックエンドログプロセス716における故障、およびシステム513の故障を処理することができる。この設計は、フォールトトレランスを与えるシステム701の2つの部分

を有し、2つの主要な目的を有する。

【0041】・パフォーマンスに関して、回復のオーバーヘッドが少ないことを保証する。
・故障および回復がアプリケーションに透過的であり、実行中のアプリケーションが停止しないことを保証する。

【0042】回復手続きは、WatchDがシステムにおける最も信頼性のある要素であるという仮定に基づく。その理由は、WatchDは非常に単純なタスクを実行し、故障後に自己回復が可能であるためである。

【0043】以下では、バックアップシステム513の故障からの回復について詳細に説明し、他のプロセスの故障からの回復についても概観する。バックアップシステム513の故障から始めると、このような場合、システム701は以下のように動作する。モニタ717は、システム513の故障を検出すると、パイププロセス711に通知する。パイププロセス711はフロントエンドログプロセス705を作成し、パイプ710のファイルディスクリプタをフロントエンドログプロセス705へのパイプ707のファイルディスクリプタで置換する。アプリケーションプロセス503によって使用されるメッセージファンクションは、パイプ710の故障を検出すると、パイププロセス711にパイプの新しいファイルディスクリプタを要求する。パイププロセス711は、フロントエンドログプロセス705に接続されたパイプ707のファイルディスクリプタをそのメッセージファンクションに与え、メッセージファンクションによって送られたメッセージは、バックエンドログプロセス716ではなくフロントエンドログプロセス705へ行く。フロントエンドログプロセス705は、そのメッ

セージを受け取ると、そのメッセージを主システム511内のログファイル703(a)に入れる。

【0044】好ましい実施の形態では、メッセージファンクションはパイプ710の故障を以下のように検出する。プロセス503はTCP/IPプロトコルを使用してパイプ710を通じてメッセージを送る。このプロトコルでは、前のメッセージが受け取られた場合に限り次のメッセージを送ることができる。従って、ライブラリルーチン507内のファンクションによって使用されるメッセージファンクションは、2つのメッセージ、すなわち、実際のメッセージおよびダミーのメッセージを送ることによってパイプ710を通じてメッセージを送る。メッセージファンクションがダミーメッセージを送ることができる場合、実際のメッセージは到着したことになる。システム513が故障すると、パイプ710を通じて送られたメッセージは到着せず、ダミーメッセージを送ることはできない。

【0045】バックアップファイルシステム513が回復すると、モニタ719は、システムコールエンジン715およびバックエンドログプロセス716を再起動し、モニタ717に通知する。モニタ717はパイププロセス711に通知し、パイププロセス711は、パイプ710のファイルディスクリプタを取得してフロントエンドログプロセス705を終了させる。バックエンドログプロセス716は、システム513において再起動されると、カーネルサーバ305(a)からログファイル703(a)のコピーを取得し、それをログファイル703(b)に付加する。続いて、システムコールエンジン715は、ログファイル703(b)内のメッセージの実行を再開する。

【0046】lib、3dによって使用されるメッセージファンクションは、パイプ707のファイルディスクリプタを取得したのと同じようにパイプ710のファイルディスクリプタを取得する。次にメッセージファンクションは、パイプ707のファイルディスクリプタを使用してメッセージを送ることを試み、この試みが失敗すると、メッセージファンクションは再びパイププロセス711にパイプファイルディスクリプタを要求する。メッセージファンクションはパイプ710のファイルディスクリプタを受け取り、再びバックエンドに接続される。

【0047】残りの故障シナリオは以下のように扱われる。

【0048】・パイププロセス711が故障した場合。モニタ717が、故障を検出し、サーバを再起動する。新たに再起動されたプロセスは、WatchDによって保存されたプロセス状態からパイプ710への接続を取得する。他のプロセスはこの故障および回復について全く知らない。

【0049】・システムコールエンジン715が故障し

た場合。

モニタ719が、故障を検出し、システムコールエンジン715を再起動する。liftによって提供されるチェックポイントおよび回復のファンクションによって、新たに再起動されたシステムコールエンジン715は、外部ファイルから、前にチェックポイントしたステータスに回復することができる。他のプロセスはこの故障および回復について全く知らない。

【0050】・バックエンドログプロセス716が故障した場合。

モニタ719が、故障を検出し、バックエンドログプロセス716を再起動する。今度も、プロセス716は、チェックポイントファイルからステータスを復元する。さらに、モニタ719は、モニタ717に、バックエンドログプロセス716が再起動されたことを通知し、続いてモニタ717は、パイププロセス711に通知する。次に、プロセス711は、パイプ710を、新しいバックエンドログプロセス716に接続する。各アプリケーションの次の書き込みは失敗し、lib. 3dはパイププロセス711から新たな接続を取得する。

【0051】・フロントエンドログプロセス705が故障した場合。

フロントエンドログプロセス705は、システム513の故障の期間中にのみ存在する。モニタ717は、フロントエンドログプロセス705の故障を検出すると、パイププロセス711に通知する。続いて、パイププロセス711は、フロントエンドログプロセス705を再起動し、それにパイプ708を再接続する。アプリケーションプログラム509の次の書き込みは失敗し、lib. 3d内のメッセージ送信ファンクションは、パイププロセス711から新たなパイプ708のファイルディスクリプタを取得する。

【0052】【ユーザレベル名前空間405の実装：図8～図11】ユーザレベル名前空間405は、カーネルサーバ305(a)によってアプリケーションプロセス503に提供されるファイルシステムからのファイルの任意のセットを指定するために使用することができる。図8に、カーネルサーバ305(a)によって提供されるファイルシステムの名前空間801と、ユーザレベルバックアップファイルシステム701内のユーザレベル名前空間405の間の関係を示す。

【0053】名前空間801において、ファイル名はツリー(木)に配置される。図8のツリーの葉をなすファイル(B, D, E, G, I, M, N)はデータまたはプログラムを含む。残りのファイルは他のファイルのリストである。このようなファイルはディレクトリと呼ばれる。名前空間801内の任意のファイルは、カーネルサーバ305(a)に対して、パス名によって指定することが可能である。パス名は、ルート「/」で始まり、ルートからそのパス名によって指定されているファイルの

名前までのすべてのファイルの名前を含む。従って、ファイルDのパス名は/A/C/Dであり、ファイルLのパス名は/J/K/Lである。

【0054】ユーザレベルバックアップファイルシステム701は、バックアップすべきファイルを、そのファイルを含む名前空間801のサブツリーを指定することによって指定する。次に、サブツリー内のファイルを変更するようなファイルに対する操作が、バックアップシステム513内のバックアップファイルに対して実行される。図8では、3つのサブツリー、803(a)、803(b)、および803(c)がバックアップすべきものとして選択されている。その結果、名前空間801内のデータファイルD、E、G、I、M、またはNへの変更の結果として、そのデータファイルに対するバックアップファイルへの変更が行われ、ディレクトリC、F、H、およびLへの変更も同様にそれらのバックアップファイルへの変更を引き起こす。サブツリー内のすべてのファイルがバックアップされるため、バックアップすべきファイルは、ユーザレベル名前空間405ではそのサブツリーのルートであるディレクトリのパス名によって指定することが可能である。こうして、サブツリー803(a)はユーザレベル名前空間405ではパス名/A/C(805(a))によって指定される。

【0055】もちろん、ユーザレベル名前空間405は、カーネルサーバ305(b)によってシステムコールエンジン715に提供されるファイルシステムにもマップされなければならない。これはバックエンドマップ517によって行われる。図9に示したように、バックエンドマップ517は、ユーザレベル名前空間405における各オープンファイルに対するエントリ901を含む。このエントリは2つの部分を有する。ユーザレベル名前空間情報903は、ユーザレベル名前空間405におけるファイルを指定し、バックアップシステム情報905は、カーネルサーバ305(b)によって提供されるファイルシステムにおいて、ユーザレベル名前空間情報によって指定されたファイルに対応するファイルを指定する。

【0056】バックエンドマップ517により、カーネルサーバ305(b)がバックエンドログプロセス716およびシステムコールエンジン715に提供するファイルシステムの名前空間907のサブツリーに、名前空間801のサブツリーをマップすることが可能となる。このマッピングは、名前空間801のサブツリーのルートのパス名を、名前空間907の対応するサブツリーのルートのパス名にマップすることによって行われる。ルートのパス名は、サブツリー内ではファイルのパス名のプレフィクスと呼ばれる。こうして、サブツリー803(a)におけるパス名はプレフィクス/A/Cを有し、サブツリー803(a)内のファイルEのパス名はEとなる。名前空間907では、名前空間801のプレフィ

クス/A/Cを名前空間907のプレフィクス/Zにマップすることによって、サブツリー909はサブツリー803(a)に対応するようになる。マッピングをした後は、名前空間801においてパス名/A/C/Eによって指定されるファイルの変更の結果、名前空間907においてパス名/Z/Eによって指定されるファイルの変更が行われることになる。

【0057】【フロントエンド複製ツリー505の詳細: 図10】好ましい実施の形態では、ユーザレベル名前空間405はフロントエンド複製ツリー505として実装される。図10に、フロントエンド複製ツリー505の詳細を示す。フロントエンド複製ツリー505の2つの主要な要素は、RTREE1015およびファイルディスクリプタ(FD) キャッシュ1027である。RTREE1015は、バックアップすべきファイルからなるサブツリー803のルートのパス名の連結リストである。ファイルディスクリプタキャッシュ1027は、ファイルディスクリプタをデバイスおよびiノード識別子に関係づける配列である。この実装の形式は、UNIXオペレーティングシステムによって提供されるファイルシステムがファイルを3通りの方法で、すなわち、パス名によって、整数のファイルディスクリプタによって、および、ファイルが存在するデバイスの識別子とUNIXファイルシステムテーブル内のそのファイルに対するエントリ(iノード)とによって、指定することの結果である。ファイルのファイルディスクリプタは、そのファイルをオープンしたプロセスに対してのみ、かつ、そのプロセスがそのファイルをオープンしている間にのみ有効である。UNIXファイルシステムテーブルでは、パス名とデバイスおよびiノードとの間、ならびに、デバイスおよびiノードと現在のファイルディスクリプタとの間の変換は可能であるが、パス名と現在のファイルディスクリプタとの間の直接の変換はできない。

【0058】さらに詳細に説明すると、MAXTRY1003およびINIT1005は、フロントエンド複製ツリー505を初期化する際に使用される。MAXTRY1003は、初期化ファンクションが、バックアップシステム513へのパイプ710の設定を試みてあきらめるまでの回数を示す。INIT1005は、パイプが設定されたかどうかを示す。RPL0P配列1009は、複製ツリー505に対して実行可能な操作の名前1011の配列である。

【0059】RTREE PTR1013は、RTREEリスト1015の第1要素へのポインタである。RTREEリスト1015は、複製ツリー803ごとに1要素1017を含む連結リストである。各要素1017は、複製ツリー803のルートのパス名1021、パス名1021の長さ1019、およびこの連結リストにおける次の要素へのポインタ1023を含む。接続サーバ1025は、バックアップシステム513へのパイプ7

10の、名前空間801におけるパス名である。

【0060】FDキャッシュ1027は、ファイルディスクリプタキャッシュエントリ1029の配列である。この配列には、アプリケーションプロセス503に利用可能なファイルディスクリプタと同じ数だけ、エントリ1029がある。FDキャッシュ1027内の与えられたファイルディスクリプタに対するエントリのインデックスはそのファイルディスクリプタである。エントリ1029は、そのエントリが現在有効であるかどうかを示し、かつ、そのファイルがオープンであった間に子プロセスを作成したかどうかをも示すステータスフラグを含む。また、エントリ1029は、主システム511においてそのファイルが存在するデバイスの識別子1101と、主システム511におけるそのファイルのiノードの識別子1103とを含む。RTREE1015内のエントリによって指定されるサブツリー803には、現在オープンのファイルごとに有効なエントリ1029が存在する。

【0061】【バックエンドマップ517の詳細】バックエンドマップ517は2つの部分、すなわち、パス名マップ1113およびオープン複製ファイルリスト1117を有する。パス名マップ1113は単に、主システム511の名前空間801内のパス名を、バックアップシステム513の名前空間907内のパス名にマップする。マップ内の各エントリ1115は、フロントエンドパス名1118とバックエンドパス名1119の間の関係を確立する。パス名マップ1113には、フロントエンド名前空間907内のサブツリー803のルートを、名前空間907内のサブツリーのルートにマップするエントリが含まれる。バックエンドパス名1119はバックエンドシステム情報905の一部である。好ましい実施の形態では、これらのマッピングは、システム設定ファイルで指定される。

【0062】オープン複製ファイルリスト1117は、アプリケーションプロセス503が複製ツリー803において現在オープンしている各ファイルに対するエントリ1120を含む。エントリ1120内のユーザレベル名前空間情報903は、フロントエンドファイル識別子(FFID)1105およびフロントエンドパス名(FP)1106を含む。フロントエンドファイル識別子1105は、主システム511内のファイルに対するデバイス識別子およびiノード識別子からなる。フロントエンドパス名1106は、フロントエンドプレフィクス(FPR)1107およびサブツリーパス名1108に分けられる。フロントエンドプレフィクス(FPR)1107は、フロントエンド名前空間801における当該ファイルのサブツリーに対するプレフィクスである。サブツリーパス名1108は、サブツリーにおけるファイルのパス名である。エントリ1120内のバックアップシステム情報905は、バックエンドファイルディスクリ

リプタ1111からなる。バックエンドファイルディスクリプタ1111は、カーネルサーバ305(b)によって提供されるファイルシステムにおける当該ファイルのファイルディスクリプタである。好ましい実施の形態では、バックエンドマップ517は、フロントエンドファイル識別子1105およびフロントエンドパス名1106のいずれによってもアクセス可能なハッシュテーブルとして実装される。

【0063】【データ構造体505および517に関する操作】以下では、どのようにしてデータ構造体505および517を作成するか、および、これらのデータ構造体がさまざまなファイル操作によってどのように影響を受けるかを説明する。好ましい実施の形態では、アプリケーションプロセス503は、Kornシェルを使用するUNIXオペレーティングシステム上で時刻される。Kornシェルによれば、プロセスは、当該プロセスがKornシェルを呼び出すときにはいつも実行されるファイルを指定するENV変数を設定することが可能である。アプリケーションプロセス503においてENV変数によって指定されるファイルは、アプリケーションプロセス503が、フロントエンド複製テーブル505を構成し初期化するのに必要な情報を含む。いったん作成されると、テーブル505は、アプリケーションプロセス503のアドレス空間の一部となり、UNIXオペレーティングシステムのforkシステムコールで作成されそれによって親の環境を継承する、アプリケーション503の任意の子プロセスに利用可能となる。他方、execシステムコールは、子プロセスに新しい環境を与える。execシステムコールで作成されるアプリケーションプロセス503の子プロセスにフロントエンド複製ツリー505が利用できるようにするため、lib、3dは、フロントエンド複製ツリー505を新しいプロセスのENV変数にコピーするexecファンクションを有する。これにより、その新しいプロセスは、親のアドレス空間を継承していなくても、フロントエンド複製ツリー505を利用することができる。他の実施の形態では、execによって作成される子プロセスにフロントエンド複製ツリー505を渡すために、名前付きパイプまたは外部ファイルを使用することも可能である。

【0064】ファイル操作の説明に進むと、第1のファイル操作はmount(マウント)操作である。UNIXオペレーティングシステムでは、mountはファイルシステムからの名前のツリーを、オペレーティングシステムの名前空間に追加する。好ましい実施の形態では、lib、3dで実装されるmountのバージョンは、フロントエンド名前空間801のサブツリーが複製ツリー805としてユーザレベル名前空間405に追加されるモードを有する。mountがこのモードで使用されるとき、パス名引数は、ユーザレベル名前空間40

5に追加されるサブツリー803のルートのパス名である。ファンクションは、そのパス名に対する複製ツリーエントリ1017を作成し、そのエントリを複製ツリー1015に追加することによって、サブツリー803をユーザレベル名前空間405に追加する。また、指定されたパス名を有する複製ツリーエントリ1017を複製ツリー1015から削除するumount(マウント解除)操作もある。

【0065】アプリケーションプロセス503が複製ツリー805内のファイルに対してオープン操作を実行すると、lib、3d内のオープンファンクションは、新たにオープンされるファイルに対するファイルディスクリプタキャッシュエントリ1029を作成し、オープンメッセージをバックエンドログプロセス716へ送る。このオープンメッセージは、オープンしたファイルの主システム511におけるパス名、デバイス識別子、およびiノード識別子を含む。このメッセージがシステムコールエンジン715によって実行されると、その結果、バックエンドマップ517内にエントリ901が作成される。パス名マップ1113を使用して、主システム511内のオープンされているファイルに対応するバックエンドシステム513内のファイルが発見され、対応するファイルに対するファイルディスクリプタがバックエンドファイルディスクリプタ1111に入れられる。

【0066】ファイルがオープンされると、主システム511におけるファイル操作は、そのファイルを識別するファイルディスクリプタを使用する。バックアップシステム513内のバックアップファイルに対する対応する操作に対するメッセージは、デバイス識別子およびiノード識別子を使用してファイルを識別する。このようなメッセージを実行するためには、システムコールエンジン715は、メッセージで指定されるデバイスおよびiノードに対するオープン複製ファイルリスト1117内のエントリ1119にアクセスするだけでよい。このエントリは、バックアップシステム513における操作を実行するのに必要なファイルディスクリプタ1111を含む。

【0067】アプリケーションプロセス503が複製ツリー505内のファイルをクローズすると、lib、3dのクローズファンクションは、ステータスフィールド1033から、子プロセスがそのファイルを使用しているかどうかを判断する。どの子プロセスも使用していない場合、クローズファンクションは、複製ツリー505内のそのファイルに対するファイルディスクリプタキャッシュエントリ1029を無効にし、デバイス識別子およびiノード識別子を含むクローズメッセージをバックアップシステム513へ送る。システムコールエンジン715は、このメッセージを実行するとき、デバイス識別子およびiノード識別子を使用してこのファイルに対するエントリ1119を見つける。続いて、このファイ

ルを識別するためにバックエンドファイルディスクリプタ1111を使用して、バックアップシステム513内のファイルをクローズし、最後に、オープン複製ファイルリスト1117からエントリ1119を削除する。

【0068】【ユーザレベルバックアップファイルシステムを使用した複製ファイルの実装：図13】バックアップファイルシステム501は、アプリケーションプロセス503のフロントエンド複製ツリー505において指定される、主システム511からの各ファイルの現在のコピーが、バックアップシステム513上に存在することが保証されるという点で有効である。しかし、主システム511からのファイルのコピーを変更するバックアップシステム513における操作の結果は、主システム511内のファイルには反映されない。実際に、このことは、主システム511しか、バックアップシステム513上にバックアップされているファイルを変更することができないことを意味する。

【0069】ファイルが複製ファイルである場合に必要とされるように、主システム511およびバックアップシステム513がいずれもファイルのコピーを変更することができるためには、各システムは、互いのシステム上でなされる変更をバックアップしなければならない。すなわち、2つのシステムは、複製ファイルのコピーに対する操作に関してピアでなければならない。図5に関していえば、2つのシステム511および513はそれぞれ、他方のシステムへのバックアップメッセージのためのチャンネル512と、バックエンドサーバ515とを有していなければならない。さらに、ファイルを変更したいシステム上のプロセスは、lib、3d(507)と、ファイルが複製ファイルとしてリストされたフロントエンド複製ツリー505を有していなければならない。さらに、複製ファイルのコピーの変更が両方のシステムにおいて同じ順序で起こること、および、複製ファイルのローカルコピーに対する読み出し操作が、複製ファイルのリモートコピーでなされた書き込みを考慮に入れて提供されることを確実にするため、同期システムが要求される。

【0070】図13に、2つのピアホスト1302

(A)および1302(B)ならびに複製ファイル1325を有する分散システム1301の概観を示す。各ホストはカーネルサーバ305(図示せず)および大容量記憶装置を有する。大容量記憶装置は、ここでは、ホスト1302(A)に対してはディスク307(a)であり、ホスト1302(B)に対してはディスク307

(b)である。各ディスクは複製ファイル1325の同一のコピーを有する。ホスト1302(A)上のコピーはコピー1325(A)であり、ホスト1302(B)上のコピーはコピー1325(B)である。さらに、各ホスト1302は、バックエンドサーバ515を有し、他方のホスト1302からバックアップメッセージを受

け取ることができる。ホスト1302(A)における3つのプロセス1309(A, 1..3)はlib、3dコード507を含み、ファイル1325を複製ファイルとして指定するフロントエンド複製ツリー505を有する。ホスト1302(B)上の1つのプロセス1309(B, 1)はコード506およびそのようなフロントエンド複製ツリーを有する。各ホスト1302は互いにバックアップとして機能するため、プロセス1309

(A, 1..3)が書き込み操作(すなわち、ホスト1302(A)上の複製ファイル1325のコピー1325(A)を変更する操作)を実行するごとに、その書き込み操作の結果、バックアップメッセージ512(A)が生じ、ホスト1302(B)上のバックエンドサーバ515(B)はこれに応答して、複製ファイルのコピー1325(B)に対して同じ書き込み操作を実行する。プロセス1309(B, 1)がコピー1325(B)に対して書き込み操作を実行すると、この書き込み操作の結果バックアップメッセージ512(B)が生じ、バックエンドサーバ515(A)はこれに応答して、コピー1325(A)に対して同じ書き込み操作を実行する。バックアップメッセージ512は、送信された順にメッセージが到着することを保証するチャンネルを通じて送られ、その結果、コピー1325(A)および1325(B)に対する書き込み操作は同じ順序で行われる。このようなチャンネルを実装する1つの方法は、TCP/IPを通じてバックアップメッセージ512を送ることである。

【0071】もちろん、バックエンドサーバ513は、プロセス1309が複製ファイル1325(A)に対して書き込み操作を実行すると同時に複製ファイル1325(B)に対して書き込み操作を実行するわけではない。その結果、ファイル1325(B)に対する読み出し操作は、ファイル1325(A)に対する同時の読み出し操作と異なる結果となる可能性がある。ある場合には、これは異なるが、他の場合には異なることがある。その結果、システム1301には、複製ファイル1325に対する2種類の読み出し操作が可能である。第1の読み出し操作は「アトミック読み出し」操作である。アトミック読み出し操作は、複製ファイル1325のコピーに一貫性がある必要がないときに使用される。この操作は単に、複製ファイル1325のローカルコピーに対する現在の書き込み操作が終了するまで待機してからそのローカルコピーを読み出す。第2の読み出し操作は「順次読み出し」操作である。この操作は、複製ファイル1325のコピーに一貫性がなければならない場合に使用され、従って、読み出されている複製ファイルのコピーは、その複製ファイルの他のすべてのコピーと一貫性があるように、複製ファイルに対する書き込み操作と同期している。

【0072】書き込み操作と順次読み出し操作の同期

は、複製ファイル1325に対する2つのトークン、すなわち、書き込みトークン1327および読み出しトークン1328によって実現される。書き込みトークン1327を有するホスト1302は、複製ファイルのローカルコピーに対する読み出し操作または書き込み操作を実行することが可能である。読み出しトークン1328を有するホスト1302は、ローカルコピーに対する読み出し操作を実行することは可能であるが、書き込み操作を実行することはできない。いずれのトークンも有しないホスト1302は、アトミック読み出し操作のみ実行可能である。ホスト1302は、必要なトークンを有しない場合、他のホスト1302にそのトークンを要求する。他のいずれかのホスト1302で書き込み操作が未完了である場合、最後の書き込みバックアップメッセージ512を送った後に、バックアップメッセージ512のために使用したチャンネルにトークンを送る。このようにトークンを送ることによって、ホスト1302は、複製ファイル1325のすべてのローカルコピーにおいて書き込み操作が同じ順序で起こること、および、複製ファイル1325の同一のローカルコピーに対して順次読み出しが実行されることを保証する。

【0073】与えられた瞬間にはただ1つのホスト1302のみが書き込みトークン1327を有し、その瞬間には、他のすべてのホスト1302はトークンを有しない。いずれのホストも書き込みトークン1327を有しない場合、すべてのホストは読み出しトークン1328を有する。読み出しトークンおよび書き込みトークンのいずれも有しないホスト1302はいずれかを要求することが可能である。書き込みトークンを有するホストは、読み出しトークンまたは書き込みトークンのいずれかを与えることが可能である。読み出しトークンを有するホストは書き込みトークンを要求または授与することが可能である。

【0074】システム1301が3つ以上のホスト1302を有するとき、書き込み操作は、書き込みトークン1327を有しないすべてのホスト1302へ同報される。トークンの要求およびトークンの授与もまたすべてのホスト1302に同報される。同報は、要求および授与の信頼性のある同報順序を提供する信頼性のある同報パッケージを使用して行われる。このようなパッケージの一例は、コーネル大学によって提供されているISISである。ISISは、「故障がある場合の信頼性のある通信(Reliable Communication in the Presence of Failures)」、ACM Transactions on Computer Systems, 5, 1, 1987年2月、第47〜76ページ、に記載されている。読み出しトークン1328の場合、読み出しトークンを授与することができる唯一のホスト1302は、書き込みトークンを有するホストである。その結果、読み出しトークンを授与するメッセージが複数存在することはない。書き込みトークン1327の場合、書

き込みトークン1327を有する単一のホスト1302が存在するか、または、すべてのホストが読み出しトークン1328を有する。前者の場合、書き込みトークン1327を授与するメッセージはただ1つ存在する。後者の場合、要求中のホスト1302は、書き込みトークン1327を実際に有する前に、読み出しトークン1328を有するすべてのホスト1302から授与メッセージを受け取らなければならない。

【0075】プロセス1309が自己のホスト1302上の複製ファイル1325のコピーに書き込みをするためには、2つの条件が満たされなければならない。

・このプロセスが実行されているホスト1302は、複製ファイルに対する書き込みトークン1327を有していなければならない。

・ホスト1302内の複製ファイル1325のコピーに対する他のホスト1302からの未完了の書き込み操作があってはならない。

【0076】ホスト1302は、書き込みトークン1327を有しない場合、他のホストに書き込みトークン1327を要求しなければならない。他のホスト1302はバックアップメッセージ512において書き込みトークン1327を送り、それによって、第2の条件が満たされること、すなわち、受信側ホスト1302は最後のバックアップメッセージ512で指定される変更が完了するまで複製ファイル1325のコピーを変更しないことを保証する。

【0077】好ましい実施の形態では、書き込みトークン1327を使用した同期は、各ホスト1302上のトークンサーバ1311と、トークンファイル1307と、バックアップメッセージ512を受信した順序で送出するチャンネルとによって実現される。トークンファイル1307は、ホスト1302上にコピーを有する各複製ファイル1325に対する領域を有する。標準的なオペレーティングシステムのロッキングサブシステムでは、ファイルの領域をロックすることが可能である。2種類のロックがある。排他ロックでは、ただ1つのプロセスのみがそのファイルにアクセスすることが可能である。共有ロックでは、任意数のプロセスがアクセスすることができる。一般に、プロセスは、領域に書き込むためにはその領域に排他ロックを有し、領域から読み出すためには共有ロックを有していなければならない。好ましい実施の形態では、トークンファイル1307における複製ファイル1325の領域に対するオペレーティングシステムロックを使用して、その複製ファイル1325に対する書き込み操作と順次読み出し操作を同期させるために使用されるトークンを実現する。

【0078】例えば、好ましい実施の形態では、書き込みトークンは、複製ファイル1325の領域に対するオペレーティングシステムロックから形成される書き込みトークンロックとして実現される。トークンサーバ13

11は、書き込みトークンを有することを示すメッセージを受け取ると、書き込みトークンロックを獲得する。トークンサーバ1311が書き込みトークンロックを有する限り、ホスト1302上で実行中のプロセス1309は、複製ファイルのローカルコピーに対するアトミック読み出し操作、順次読み出し操作、または書き込み操作のロックを獲得することができる。これらのロックもまた、トークンファイル1307におけるオペレーティングシステムロックを使用して実現される。

【0079】他のホスト1302が書き込みトークンを要求すると、トークンサーバ1311は書き込みトークンロックを解放し、トークンなしロック（他のロックと同様に実現される）を獲得する。トークンサーバ1311がトークンなしロックを有する限り、ホストシステム1302において複製ファイル1325に書き込むことが可能な唯一のプロセスはバックエンドサーバ515である。もちろん、バックエンドサーバ515は、現在書き込みトークン1327を有するホスト1302からのバックアップメッセージ512に応答する。

【0080】システム1301の動作は以下の通りである。ユーザレベルバックアップファイルシステム501の説明で既に述べたように、lib. 3d (507)は、複製ファイルに対する操作を実行するアプリケーションプロセス509のコードに静的にまたは動的にバインドされる。その後、ファイルがフロントエンド複製ツリー505において複製ファイルとして指定される。システム1301で使用されるlib. 3d (507)のバージョンは、標準的なI/Oライブラリ書き込みルーチンを、図12に示す書き込み操作で置き換える。第3行のget_write_token()関数1201は、関連するホストのトークンサーバ1311に書き込みトークン1327を要求する。そのホストのトークンサーバ1311が書き込みトークン1327を有する場合、この関数は直ちに復帰する。トークンサーバ1311は、書き込みトークン1327を有していない場合、他のホストに要求し、書き込みトークンが到着すると復帰する。トークンサーバ1311が書き込みトークン1327を有すると、プロセス1309は第4行で書き込みシステムコールsyscall(SYS_write, fildes, buf, nbytes)を実行する。その後、システム501の説明で述べたように、関数は、フロントエンド複製ツリー505から、ファイルが複製されているかどうかを判断する。複製されている場合、書き込みメッセージ512が他のホスト1302に送られ、書き込みトークンは解放される(1203)。書き込みトークン1327は、同じようにして、複製ファイル1325を変更するいずれのホスト1302上の書き込み操作に対しても、獲得されなければならない。その結果、すべての変更は複製ファイル1325のすべてのコピーに対して行われ、すべての変更は同じ順序で行われる。

【0081】[同期の詳細な実装: 図14] 好ましい実施の形態では、与えられたホスト1302上の複製ファイル1325のコピーに属するロックファイル1307の領域は、書き込み操作に関連する2つのロックを有する。第1のロックは、書き込みトークン1327がホスト1302上にあるかどうかを示し、第2のロックは、複製ファイル1325のコピーが、当該ホスト1302上のプロセス1309による書き込みに利用可能であるかどうかを示す。図14に、好ましい実施の形態においてこれら2つのロックをどのようにして使用するかを示す。図の擬似コード1401は今度もlib. 3d (507)の書き込み操作に対するものである。ロックを含む領域は変数TOKEN_REGION (1403)によって表され、これは2つのフィールドを有する。ロックのSTATEは、書き込みトークン1327がホスト1302上にあるかどうかを示し、ロックのTOKENは、プロセス1309が書き込みを実行することができるかどうかを示す。STATEによって表されるロックは、トークン1327が他のホスト1302上にあるとき、ローカルトークンサーバ1311によって排他ロックされたまま保持される。

【0082】擬似コード1401によって記述される動作は以下の通りである。第3行に示されるように、ロック1403を含むトークンファイル1307の領域が関数fd2tokenによって検索される。この関数は、複製ファイル1325のローカルコピーのファイルディスクリプタをとり、領域1403を返す。次のステップで、複製ファイル1325に対する書き込みトークン1327がローカルホスト1302内にあるかどうかを判定する。これは、第4行で、領域1403のSTATEフィールドの非ブロッキング共有ロックを要求することによって行われる。このロックが取得可能である場合、書き込みトークン1327はローカルホスト1302上にある。このロックが取得可能でない場合、擬似コード1401は、トークンサーバ1311が他のホスト1302上の対応するトークンサーバにトークン1327を要求するメッセージを送り、そのトークンを提供するメッセージが返るのを待機するようにする関数(図示せず)を呼び出す。第4行では、書き込みトークン1327がローカルホスト1302上にあるかどうかを複数のプロセス1309が判定できるように、共有ロックの取得を試みる。

【0083】書き込みトークン1327がローカルに利用可能となると、次のステップへ進む。第6行に示されるように、領域1403のSTATEフィールドに対してもう1つのロック要求がなされる。今度はこれはブロッキングであり、コード1401を実行しているプロセス1309は、STATEに対する共有ロックを取得することができるまで（すなわち、書き込みトークン1327がローカルに利用可能になるまで）待機し（第6行）、その後、TOKENに対する排他ロックを獲得するまでブロック

する。プロセス1309が排他ロックを受け取ると、実際に書き込みシステムコールがなされ、メッセージ512が、複製ファイル1325のローカルコピーへの書き込みの内容とともに、他方のホスト1302へ送られる。これが行われると、領域1403はロック解除され、書き込み操作が終了する。

【0084】もちろん、ローカルホスト1302上で複製ファイル1325に対する書き込みを試みている他のいずれのプロセス1309も、STATEに対する共有ロックを有し、TOKENに対する排他ロックを待機しているプロセスの待ち行列に入ることができる。書き込みを完了したプロセス1309がTOKEN_REGIONをロック解除すると、待ち行列における次のこのようなプロセスがTOKENに対する排他ロックを取得し書き込み操作を実行することができる。さらに、ローカルトークンサーバ1311が、他のトークンサーバ1311から複製ファイル1325に対する書き込みトークン1327を要求するメッセージ512を受け取ると、ローカルトークンサーバ1311はSTATEに対する排他ロックを要求する。ローカルトークンサーバ1311は、STATEに対する共有ロックを有するすべてのプロセス1309が書き込み操作を完了した後にのみ、その排他ロックを受け取る。ローカルトークンサーバは、STATEに対する排他ロックを受け取ると、書き込み操作によって生成されたメッセージが他のホスト1302に送られたのと同じチャネルによって、そのことを知らせるメッセージを他のホスト1302へ送る。チャネルに入れられたメッセージは、送られた順序で到着し、その結果、他のホスト1302上のトークンサーバ1311は、当該他のホスト1302上のバックエンドサーバ515が、トークン1325を有していたホスト1302からのすべての書き込みメッセージ512を処理した後にのみ、STATEに対する排他ロックを解放する。

【0085】他のホスト1302上で実行される順次読み出し操作と書き込み操作の同期は、書き込みトークン1327に関して説明したのとほぼ同様に、読み出しトークン1328によって達成される。順次読み出しを実行するプロセス1309は、読み出しトークン1328または書き込みトークン1327がホスト1302上にあるかどうかを示すトークンファイル1307の一部に対する共有ロックを取得することをまず試みる、1 i b. 3 d (507) 内のコードを実行する。この試みが失敗した場合、プロセス1309は、トークンサーバ1311が読み出しトークン1328を他のホスト1302から取得することを要求する。トークンは、書き込みトークン1327について説明したのと同様にして取得される。次に、プロセス1309は、そのトークンを表す領域に対する排他ロックを取得することを試み、読み出しトークン1328がホスト1302に到着するまでブロッキングする。トークンが到着すると、プロセス1

309は、複製ファイル1325のローカルコピーに対する共有ロックを要求する。プロセス1309は、そのローカルコピーが他のプロセス1309によって実行されているローカル書き込み操作に対する排他ロックでなく、かつ、バックアップメッセージ512に回答してバックアップサーバ515によって実行されているリモート書き込み操作に対する排他ロックでもない場合にのみ、その共有ロックを受け取ることができる。

【0086】既に示したように、書き込みトークン1327を有するトークンサーバ1311は読み出しトークン1328を授与することができる。書き込みトークン1327を有するトークンサーバ1311が要求を受け取った場合、複製ファイルに対する書き込み操作が終了するのを待ち、複製ファイルのローカルコピーに対するロックを排他ロックから共有ロックに変更し、書き込みバックアップメッセージ512のために使用しているチャネルによって読み出しトークンを送る。このことすべてにより、読み出しトークン1328は、最後の書き込みバックアップメッセージ512の後に、要求側ホスト1302に到着する。

【0087】[複製ファイルに対する状態マシン：図15] ホスト1302内のアプリケーションプロセス1309、トークンサーバ1311、およびバックエンドサーバ515の協力ならびにホスト1302間でのトークンサーバ1311の協力は、プロセス1309、トークンサーバ1311、およびバックエンドサーバ515を状態マシンとして考察することによってより良く理解される。与えられた複製ファイル1325に関して、これらの各マシンの状態は、そのファイルに対する書き込みトークン1327および読み出しトークン1328に依存し、トークンがホスト1302間を移動するにつれて変化する。

【0088】アプリケーションプロセス1309は、複製ファイル1325に関して4つの状態を有する。

【0089】1. 操作なし状態。この状態では、アプリケーションプロセス1309は、複製ファイル1325のローカルコピーに対するいかなる種類のロックも有しておらず、従って、ローカルコピーに対する読み出し操作も書き込み操作を実行することができない。

【0090】2. アトミック読み出し状態。この状態では、プロセス1309はローカルコピーに対する共有ロックのみを有し、従って、アトミック読み出し操作のみを実行することができる。

【0091】3. 順次読み出し状態。この状態では、複製ファイル1325に対する読み出しトークン1328または書き込みトークン1327がホスト1302にあり、プロセス1309はローカルコピーに対する共有ロックを有し、従って、アトミック読み出しのみならず順次読み出し操作を実行することができる。

【0092】4. 書き込み状態。この状態では、複製フ

ファイル1325に対する書き込みトークン1327がホスト1302にあり、プロセス1309はローカルコピーに対する排他ロックを有し、従って、順次読み出しおよびアトミック読み出し操作のみならず書き込み操作を実行することができる。書き込み操作は、複製ファイル1325の他のコピーでバックアップされる。

【0093】状態の説明から明らかなように、ある状態から他の状態への遷移には、トークンおよびロックの獲得および喪失が伴う。例えば、操作なし状態から順次読み出し状態への遷移は、ホスト1302における読み出しトークン1328の獲得およびプロセス1309による共有ロックの獲得を要求する。

【0094】バックエンドサーバ515を実現するプロセスの状態は、アプリケーションプロセス1309の状態1および4と密接に関連している。

【0095】1. 操作なし状態。この状態では、バックエンドサーバ515は、複製ファイル1325のローカルコピーに対するいかなる種類のロックも有しておらず、従って、ローカルコピーに対する読み出し操作も書き込み操作を実行することができない。

【0096】2. 書き込み状態。この状態では、バックエンドサーバ515は、複製ファイル1325のローカルコピーに対する排他ロックを有し、従って、ローカルコピーに書き込みをすることができる。

【0097】以上のことからわかるように、状態変化は、ホスト1302間のトークンの移動によって引き起こされる。

【0098】最も複雑な場合は、トークンサーバ1311 (A) および (B) を実現するプロセスの場合である。トークンサーバ1311は、ホスト1302 (A) と (B) の間でトークンを渡すために相互に協力しなければならない。図15は、2つのトークンサーバ1311に対する状態図である。図15において、各状態は番号1501、1502、...、1506を有する。状態遷移は矢印で示される。矢印の参照番号の最後の2桁は、その矢印で示される遷移がなされる始状態および終状態を示す。従って、矢印1531は、状態1503から状態1501への遷移を示す。各状態遷移は、トークンサーバ1311で受け取られるメッセージの結果であり、これにより、メッセージが他のトークンサーバ1311に送られることもある。

【0099】図15において、与えられた遷移に対して受け取られるメッセージは、その遷移の矢印のそばに斜体字で示され、送られるメッセージはブロック体で示される。例えば、矢印1531で示される遷移は、他のトークンサーバ1311からのTSgetRtokenTS (斜体) メッセージの結果である。また、この遷移は、他のトークンサーバへのTSgrantRtokenRP_TS (ブロック体) メッセージを生成する。メッセージの名前は、メッセージの始点、宛先、内容、およびそのメッセージがとる経路を示

す。例えば、TSgrantRtokenRP_TS (ブロック体) は、バックアップメッセージ512のために使用されるチャンネルを通じて送られなければならない (RP_)、一方のトークンサーバ (第1のTS) から他方のトークンサーバ (第2のTS) への読み出しトークン授与 (grantRtoken) メッセージである。同様に、TSgetRtokenTS (斜体) は、一方のトークンサーバから他方のトークンサーバへの読み出しトークン要求メッセージであるが、このメッセージは、バックアップメッセージ用のチャンネルを通じて送られる必要はない。

【0100】図15の概観からはじめると、まず、3つの主要な状態がある。

【0101】・書き込みトークン状態1503。この状態では、ローカルホスト1302は書き込みトークン1327を有し、プロセス1309は複製ファイル1325のローカルコピーに対するすべての読み出し操作および書き込み操作を実行することが可能であり、トークンサーバ1311は読み出しトークンおよび書き込みトークンの両方を授与することが可能である。

【0102】・読み出しトークン状態1501。この状態では、ローカルホスト1302は読み出しトークンのみを有し、プロセス1309は複製ファイル1325のローカルコピーに対するすべての読み出し操作を実行することができるがローカルコピーへの書き込みはできず、トークンサーバ1311は書き込みトークン1327の要求または授与をすることができる。

【0103】・トークンなし状態1505。この状態では、ローカルホスト1302はトークンを有さず、プロセス1309は複製ファイル1325のローカルコピーに対するアトミック読み出し操作のみを実行することが可能であり、バックエンドサーバ515のみが複製ファイル1325のローカルコピーに書き込むことが可能であり、トークンサーバ1311は、読み出しトークンまたは書き込みトークンの要求のみをすることが可能である。

【0104】図15における始状態は読み出しトークン状態1501である。アプリケーションプロセス1309が読み出しトークン状態1501の間に書き込み操作を試みると、状態遷移1512が起こる。トークンサーバ1311はプロセス1309からAPgetWtokenTS (斜体) 要求を受け取り、この要求に応答して、他のホスト1302のトークンサーバ1311へTSgetWtokenTS (ブロック体) メッセージを送る。ここで、ローカルトークンサーバ1311は、読み出しトークン・書き込みトークン待機 (RwaitW) 状態1502において、他のトークンサーバから書き込みトークンを授与するTSgrantWtokenTS (斜体) メッセージを受け取るまで待機する。状態の名前が示しているように、ローカルホストは、書き込みトークンを待っている間、読み出しトークンを保持する。TSgrantWtokenTS (斜体) メッセージを受け取

ると、遷移1523が起こり、トークンサーバ1311は書き込みトークン状態1503に入り、ローカルホスト1302に書き込みトークン1327が来る。ここで、書き込みトークンに対する要求を生じた書き込み操作が実行され、その結果、複製ファイル1325のローカルコピーに書き込みが行われ、他のホスト1302へ書き込みバックアップメッセージ512が送られる。

【0105】もちろん、いずれのトークンサーバ1311も書き込みトークン1327を有さず、それぞれが他方から書き込みトークン1327を要求する可能性もある。その場合、状態1502においてタイブレークアルゴリズム(状態遷移1522によって表されている)が実行され、いずれのホスト1302が書き込みトークン1327を受け取るかが決定される。このアルゴリズムは、一方のホストが一次ホストとして指定され、他方のホストが二次ホストとして指定されることを要求する。一次ホストは、他方のホスト1302からの要求に回答して、その要求を無視して状態1503を続行する。二次ホストはトークンなし・書き込みトークン待機(Nwaitw)状態1504への遷移1524を行う。

【0106】トークンサーバ1311は、状態1503にあるとき、書き込みトークン1327を有し、要求中のトークンサーバ1311へ書き込みトークン1327または読み出しトークン1328のいずれかを提供することができる。読み出しトークン1328に対する要求に回答して、矢印1531で示される遷移が起こる。TSgetRtokenTS(斜体)メッセージに回答して、トークンサーバ1311は書き込みトークン1327を放棄し、要求中のトークンサーバ1311へTSgrantRtokenRP_TS(ブロック体)メッセージを送るが、読み出しトークン1328は保持する。その結果、いずれのトークンサーバ1311も読み出しトークンを有し、状態1501にあることになる。

【0107】要求が、書き込みトークン1327に対するものであるとき、矢印1535で示される遷移が起こる。TSgetWtokenTS(斜体)要求に回答して、トークンサーバ1311は自己のトークンを放棄し、メッセージ512のために使用されるチャンネルを通じてTSgrantWtokenRP_TS(ブロック体)メッセージをバックエンドサーバへ送り、トークンサーバ1311は状態1505に入る。状態1505は、状態1501から遷移1515によって到達することもある。この遷移は、状態1501にあるトークンサーバ1311がTSgetWtokenTS(斜体)メッセージを受け取り、それに回答して今遷移1535について説明したように動作するとき起こる。ただし、トークンサーバ1311のホストは書き込みをしているのではないので、メッセージ512のチャンネルを通じてバックエンドサーバへメッセージを送る必要はない。

【0108】ローカルホスト1302上のアプリケーシ

ョンプロセス1309が複製ファイル1325のローカルコピーに対する読み出し操作または書き込み操作を実行しようと試みるまで、トークンサーバ1311は状態1505にとどまる。読み出し操作の場合、この試みの結果、遷移1556が起こり、プロセス1309からのAPgetRtokenTS(斜体)メッセージがトークンサーバ1311によって受け取られ、トークンサーバ1311はこれに回答して他方のトークンサーバ1311へTSgetRtokenTS(ブロック体)メッセージを送る。次に、トークンサーバ1311は、トークンなし・読み出しトークン待機(NwaitR)状態1506に入り、読み出しトークン1328を待機する。待機中、書き込みトークン1327に対するローカル要求は待ち行列に入れる。このトークンを授与するTSgrantRtokenRP_TS(斜体)メッセージがメッセージ512のチャンネルを通じて到着すると、結果として、読み出しトークン状態1501への遷移1561が起こる。

【0109】アプリケーションプロセス1309が複製ファイルに対する書き込み操作を試みた場合、結果として遷移1554が起こる。この遷移において、トークンサーバ1311はAPgetWtokenTS(斜体)メッセージに回答して、TSgetWtokenTS(ブロック体)メッセージを他方のトークンサーバ1311へ送り、その結果、状態1504に入る。次に、トークンサーバは、状態1504において、他方のトークンサーバ1311からのTSgrantWtokenRP_TS(斜体)メッセージを待機する。このメッセージが送られるチャンネルは、バックエンドサーバ515へのメッセージ512のためのものである。TSgrantWtokenRP_TS(斜体)メッセージが到着すると、状態1503への遷移1543が起こる。

【0110】[同期の実装:図1、図16、図17] 好ましい実施の形態では、同期は、複製ファイル1325の各ローカルコピーに対する7個のロックのセットによって実装される。図16に、ロックの種類、ロックを有しなければならないシステム1301の要素、および、ロックの意味のリストである。ロック1601、1603、および1605は、それぞれ、プロセス1309が複製ファイル1325のローカルコピーに対してアトミック読み出し操作、順次読み出し操作、および書き込み操作を実行するために有しなければならないロックである。ロック1607は、ローカルバックエンドサーバ515が複製ファイル1325のローカルコピーに書き込みをするために有しなければならないロックである。ロック1609、1611、および1613は、現在ローカルホスト1302にある複製ファイル1325に対するトークンによって要求されるように、ローカルトークンサーバ1311によって要求される。例えば、ローカルホスト1302が書き込みトークンを有し、その書き込みトークンに対する要求を受け取った場合、ローカルトークンサーバ1311は、書き込みトークンロック1

609を解放し、トークンなしロック1613を獲得する。

【0111】図17に、ロックの意味規則を示す。図16の各ロックに対する行および列がある。行と列の交点にxがある場合、相異なる要求者がその行のロックとその列のロックとを同時に保有することはできない。例えば、トークンサーバ1311がトークンなしロック1613を保有している場合、ローカルホスト1302にトークンがないという状況に対して要求されるとおり、プロセス1309は、順次読み出しロック1603またはローカル書き込みロック1605を有することはない。

【0112】好ましい実施の形態では、図16のロックは、UNIXオペレーティングシステムのSunOSオペレーティングシステムまたはSystem V Release 4のようなオペレーティングシステムによって提供される共有ロックおよび排他ロックにより実装される(SunOSはSun Microsystems, Inc.の商標である)。オペレーティングシステムによって提供されるロックにより、プロセスは、ファイルのバイトに対する共有ロックまたは排他ロックを取得することができる。さらに、このバイトは、他のファイルに対するロックを表すために使用することも可能である。こうして、好ましい実施の形態では、図16のロックは、トークンファイル1327において、トークンファイル1327内のバイト列を各複製ファイル1325に割り当て、複製ファイルのバイト列内のバイトを使用して複製ファイルのロックを表現することによって実装される。複製ファイル1325に対するバイト列はスロットと呼ばれる。好ましい実施の形態では、各スロットは3バイトを有する。図16の各ロックはスロットのバイトに対するロックの組合せによって表現される。

【0113】図1に、好ましい実施の形態において使用される組合せを示す。第1列は、システム1301内の複製ファイル1325のローカルコピーに対して使用されるロックのリストである。第2列は、どの種類のオペレーティングシステムロックが使用されるかを示す。Rは共有ロック、Wは排他ロック、およびNLはロックなしを示す。残りの列は、スロットのバイトを示す。バイトごとの列内のダッシュは、第2列で指定されるOSロックがそのバイトに対して獲得されていることを示す。こうして、プロセス1309は複製ファイル1325のローカルコピーに対するアトミック読み出しロック1601を獲得しているとき、オペレーティングシステムは複製ファイル1325に対するスロットのバイト0に共有ロックを有する。同様に、プロセス1309がローカルコピーに対する書き込みロック1605を獲得しているとき、オペレーティングシステムはスロットのバイト1および2に共有ロックを有し、バイト0に排他ロックを有する。

【0114】OSロックのこのマッピングは、システム

1301のロックに対する図17の衝突テーブルを実現する。衝突しているシステム1301のロックは、衝突しているOSロックおよび重複したオフセットにマップされ、一方、衝突していないシステム1301のロックは、衝突していないOSロックまたは重複しないオフセットにマップされる。好ましい実施の形態では、このマッピングはSLEVEと呼ばれる同期ツールによって自動的に生成される。(エイ、スカーラ(A. Skarra)、

「SLEVE: イベント同期のための意味規則ロック(SLEVE: Semantic Locking for Event synchronization)」、Proceedings of Ninth International Conference on Data Engineering (1993年) 参照。)

【0115】【複製ファイルに対する高水準操作の実行】複製ファイルの各ローカルコピーが同様のすべての他のコピーと等価であるということの重要な結果として、上記の書き込み操作と全く同様にして高水準の操作を扱うことができる。書き込み操作では、書き込みトークン1327を有するホスト1302が複製ファイル1325のローカルコピーに対する書き込みを実行し、その後、その書き込みおよび書き込まれるデータを指定するメッセージを他のホスト1302へ送り、そこで、バックエンドサーバ515が、メッセージに指定された書き込み操作を、当該他のホスト1302内の複製ファイル1325のローカルコピーに対して実行する。全く同じことを高水準操作、例えば、2つの複製ファイルに関するソート(整列)・マージ(併合)を行う場合にも行うことができる。複製ファイルはすべてのホスト1302において等価であるため、書き込みトークン1327を有するホスト1302は以下のように進むことが可能である。ソート・マージを行いその操作に伴うすべての書き込みに対する書き込みバックアップメッセージ512を送る代わりに、ホスト1302は、ローカルコピーに対してソート・マージを実行してから、そのソート・マージ操作を指定するメッセージ512を他のホスト1302へ送ることが可能である。複製ファイル1325は他のすべてのホスト1302上で等価であるため、この指定されたソート・マージ操作の結果はすべてのホスト1302上で同一となる。このアプローチの利点は、ソート・マージ操作の指定を送ることは、書き込みトークンを有するホスト1302上のソート・マージの結果生じるすべての書き込み操作を他のホスト1302へ送るよりもずっと少ない時間およびネットワーク資源しか必要としないことである。

【0116】さらに詳細に説明すると、高水準操作は以下のような状況で使うことができる。各ホスト1302が2つの複製ファイル、すなわち、ソートされたマスタリストファイルおよび更新ファイルを有する。マスタリストファイルへの更新は更新ファイルに対して行われ、ソート・マージ操作が更新ファイルおよびマスタリストファイルに対して周期的に実行されて新たなマスタ

リストファイルが生成される。その後更新ファイルは削除され、再作成されて、このサイクルが再開される。更新ファイルに対する更新は、上記のような書き込み操作を使用して行われる。従って、ソート・マージ操作の時刻が来ると、すべてのホスト1302は同一の更新ファイルおよびマスタリストファイルを有する。書き込みトークンを有するホスト1302はそれ自身ソート・マージ操作を実行してから、書き込みバックアップメッセージ512を送る場合と全く同様にしてソート・マージ操作の指定を有するメッセージ512を他の各ホスト1302へ送る。この指定は、ソート・マージ操作のコードであることも可能であるが、この操作は反復して実行されるため、この指定は一般には、ソート・マージが更新ファイルおよびマスタリストファイルに対して実行されることを指定するコマンドラインである。次に、ソート・マージが他の各ホスト1302に対して実行される。同様に進行して、書き込みトークンを有するホスト1302は更新ファイルを削除し再作成して、同じ操作を指定するメッセージを他のホスト1302へ送る。もちろん、操作の指定は任意のレベルのものが可能である。例えば、上記の例では、更新ファイルのソート・マージならびに削除および再作成をすべてシェルスクリプトにおいて指定することが可能であり、操作指定は、そのシェルスクリプトとするか、または、他のすべてのホスト1302がそのシェルスクリプトのコピーを有する場合には、そのシェルスクリプトを呼び出すために使用するコマンドラインとすることも可能である。

【0117】【複製ファイルへのアクセスのトランザクション的同期】説明したように、複製ファイルシステムは、ファイルへの単一のアクセスに対する複製ファイル間の一貫性を保証する。また、複製ファイルシステムは、ファイルへのトランザクション的アクセスに対する一貫性を保証するためにも使用することができる。トランザクションとは、単一アクセスのシーケンスからなる単一の論理アクションである。トランザクションの例は以下の通りである。ファイルf内の各レコードrは、プログラムPがrを読み出した回数を記憶する属性readnumを含むと仮定する。Pは以下の擬似コードにおいてreadnumをインクリメントする。ただし、関数のパラメータリストは省略してある。

```
【0118】 for every r in f
read(f); increment_readnum(); write(f);
```

【0119】Pのいくつかのインスタンスが並行して実行されるときには、アクセスレベルの同期だけではfのレコードにおける一貫性を保証するのに十分ではない。あるPがrを読み出した直後は、Pがreadnumをインクリメントしrをfに書き込むまでは、他のプログラムはrの読み出しまたは書き込みを行うべきではない。読み出し-インクリメント-書き込みのシーケンスは、アトミック単位として同期すべきrに対する単一の論理アク

ション（すなわち、トランザクション）である。

【0120】別個のトランザクション機構がない場合、プログラマは、オペレーティングシステムのロックプリミティブを用いてこのようなシーケンスに対するトランザクション同期の孤立性を実装することができる。プログラマは単にシーケンスをexclusive_lock(f)...unlock(f)のようなプリミティブでくるだけである。一般の場合、計算は複数のファイルに関係し、プログラムは、孤立性を保証するときに複数のロックを要求しなければならない。例えば、計算が2つのファイルf1およびf2の内容に依存し、一方、その結果は他のファイルf3に記憶される。

```
read(f1); read(f2); computation(); write(f3);
そして、プログラムは以下のようにこのシーケンスをロック要求で囲む。
```

```
share_lock(f1); share_lock(f2); exclusive_lock(f3) ...
```

```
unlock(f3); unlock(f2); unlock(f1);
```

しかし、プロセスが一時に複数のロックを要求すると、デッドロックが起こる可能性がある。同期プロトコルを、トランザクション機構によって与えるか、または、オペレーティングシステムロックを使用するアプリケーションによって定義するかによって、デッドロックを防止するかまたは検出して解決するかしなければならない。

【0121】既存のオペレーティングシステムはローカルエリアネットワーク内の（リモート）ファイルをロックするプリミティブをサポートしている。しかし、分散環境では、高い通信コストのためにロックは効果であり、特にデッドロック検出は大域的アルゴリズムを必要とするため、より困難である。現在利用可能なオペレーティングシステムのロックプリミティブは、分散環境におけるトランザクション的同期を十分にサポートしていない。

【0122】ここで説明した複製ファイルシステムは、ローカルエリアネットワークまたは広域ネットワークにわたる複製ファイルにアクセスするプログラムシーケンスのデッドロックのないトランザクション的同期をサポートするTXというサービスを提供する。複製ファイルシステムは、並列プロセスを同期させるために必要なメッセージの数を最小にするプロトコルを実現する。

【0123】好ましい実施の形態では、複製ファイルシステムは、管理する各ファイルfごとに現シーケンス番号(curSeqNf)を生成する。複製ファイルシステムは、プロセスが複製ツリー505にfを作成するとき（またはこの複製ツリー505にfを移動するとき）にcurSeqNfを0に初期化し、プロセスがwrite()操作でfを変更するときcurSeqNfをインクリメントする。プロセスPは、fに対する読み出しトークンまたは書き込みトークンを獲得することの一部としてcurSeqNfを受け取る。こ

の値は大域的（グローバル）である。すなわち、curSeqNfは、f への変更に係るプロセスがPに対してローカルであるかリモートであるかにかかわらず、そのすべての変更を反映する。

【0124】TXを使用するために、プロセスPはフラグ0_TXで複製ファイルfをオープンする。Pが最初にfの読み出しまたは書き込みをするとき、TXはcurSeqNfを局所変数locSeqNfに保存する。Pが次にfの読み出しまたは書き込みをするとき、TXはlocSeqNfをcurSeqNfと比較する。これらが等しくない場合、その間に他のプロセスがfを変更したことになり、TXはエラーを返す。そうでない場合、TXはオペレーティングシステムのreadまたはwriteファンクションを呼び出し、その結果を返す。

【0125】複製ファイルシステムは、デッドロックを避けるために、複製するファイル全体の順序を定義する。TXファンクションが複数のトークンを要求するとき、事前に定義された順序に従ってその要求を行う。

【0126】【インタフェース】TXの制御構造は、UNIXインタフェースopen/close/read/writeの名前および返値を保ったまま再定義し拡張する関数インタフェース内にカプセル化される。TXは、errnoに対する新たな値としてETXを定義する。これは、トランザクションの同期によるエラーを意味する。また、TXは2つの新たな関数readtx()およびwritetx()を定義する。これらは、ファイルの集合にアクセスし、型fd_set*（システムコールselectの場合と同様）およびtx_t*のパラメータを有する。ただし、struct tx {char *buf; int nbyte;}である。プログラムは、読み出しまたは書き込みをするファイルに対してfd_setにセットされたビットにより関数を呼び出し、ETXエラー復帰の場合、関数は、fd_setにおいて、curSeqNfが変化しているファイル以外のすべてのビットをクリアする。インタフェースおよび擬似コードは以下の通りである。

【0127】・open(char *path; int flags; mode_t mode)

flagsが0_TXを含む場合、局所変数locSeqNpathを0に初期化し、0_TXをflagsから削除する。システムコールopenを呼び出し、その結果を返す。

【0128】・read(int fd; char *buf; int nbyte)
fdでオープンしたファイルfに対する読み出しトークンを取得する。fdに対して0_TXがセットされていない場合、システムコールreadを呼び出し、読み出しトークンを解放し、復帰する。それ以外の場合、locSeqNf=0であれば、curSeqNfをlocSeqNfに代入する。locSeqNf≠curSeqNfの場合、errno = ETXとセットし、そうでない場合、システムコールreadを呼び出す。読み出しトークンを解放し復帰する。

【0129】・write(int fd; char *buf; int nbyte)
fdでオープンしたファイルfに対する書き込みトークンを取得する。fdに対して0_TXがセットされていない場

合、fのローカルコピーに対するシステムコールwriteを呼び出し、それが成功した場合、その更新をリモートコピーへ送り、書き込みトークンを解放し、復帰する。それ以外の場合、locSeqNf=0であれば、curSeqNfをlocSeqNfに代入する。locSeqNf≠curSeqNfの場合、errno = ETXとセットし、そうでない場合、上記のようにfのコピーを更新し、0_TXの場合、locSeqNf（およびcurSeqNf）をインクリメントする。書き込みトークンを解放し復帰する。

10 【0130】・readtx(fd_set *readfds; struct tx_t *txp)

セットされた各ビットreadfds[fd]に対して、複製ファイルシステムによって定義される順序で、fdでオープンされたファイルfに対する読み出しトークンを取得し、fdに対して0_TXがセットされている場合、次のことを実行する。locSeqNf=0の場合、fを_nullとマークし、それ以外の場合、locSeqNf≠curSeqNfであれば、fを変更ありとマークする。いずれかのfが変更ありの場合、errno = ETXとセットし、*readfdsにおいて、変更されたファイルに対するビット以外のすべてのビットをクリアする。いずれのfにも変更がない場合、各ファイルfに対して、システムコールreadを呼び出し、fが_nullである場合にはcurSeqNfをlocSeqNfに代入する。すべてのトークンを解放し復帰する。この関数は、いずれかのファイルが変更ありの場合にはいずれのファイルも読み出さず、それ以外の場合にはすべてのファイルを読み出す。

20 【0131】・writetx(fd_set *depends_on, *writefds; struct tx_t *txp)

30 セットされた各ビットdepends_on[fd]またはwritefds[fd]に対して、それぞれ、複製ファイルシステムによって定義される順序で、fdでオープンされたファイルfに対する読み出しトークンまたは書き込みトークンを取得し、fdに対して0_TXがセットされている場合、次のことを実行する。locSeqNf≠0かつ≠locSeqNf≠curSeqNfである場合、fを変更ありとマークする。いずれかのfが変更ありの場合、errno = ETXとセットし、*depends_onまたは*writefdsにおいて、変更されたファイルに対するビット以外のすべてのビットをクリアする。いずれのfにも変更がない場合、*writefds内の各fdに対してシステムコールwriteを呼び出す。これが成功した場合、fのリモートコピーへその更新を送り、0_TXがセットされている場合、locSeqNfをcurSeqNf+1とセットする（そしてcurSeqNfをインクリメントする）。すべてのトークンを解放し復帰する。この関数は、いずれかのファイルが変更ありの場合には書き込みを実行せず、それ以外の場合にはすべての書き込みを実行する。

【0132】・resettx(fd_set *fds)

50 セットされた各ビットfds[fd]に対して、fdでオープンされているファイルfに対するlocSeqNfを0に再初期化

する。

【0133】 [使用法] TXはアプリケーションが定義する再試行プロトコルをサポートする。例示のために、新しい各レコードとの新しいトランザクションを開始する*

```
if(fd = open(f, O_RDWR O_TX)) < 0
    exit;
FD_ZERO(&fdset);
FD_SET(fd, &fdset);
for every r in f
    resettx(&fdset);
    for (try = TRY_NUM; try > 0; try--) {
        if read(fd, buf, nbytes) < 0
            exit;
        increment_readnum();
        if write(fd, buf, nbytes) >= 0
            break;
        if errno != ETX
            exit;
        /* そうでなければリセットし再試行する。*/
        /* いずれか他のプロセスがread()以降にfを変更した。*/
        resettx(&fdset);
    }
```

【0134】 プログラムは、新たな読み出し—インクリメント—書き込みのシーケンスを開始するごとにresettx()でlocSeqNfを再初期化する。これは、このシーケンスが、fの(前の)状態に依存しない単一の論理アクションから構成されるためである。

【0135】 これに対して、ファイルf1、f2および※

```
/* O_TXフラグとともに、fd1~fd3でf1~f3をオープンする。*/
/* バッファでtx_t配列を初期化する。*/
FD_ZERO(&readset);
FD_ZERO(&writeset);
for (try = TRY_NUM; try > 0; try--) {
    FD_SET(fd1, &readset);
    FD_SET(fd2, &readset);
    if readtx(&readset, txp) < 0 {
        if errno != ETX
            exit;
        resettx(&readset);
        continue;
    }
    Computation();
    FD_SET(fd3, &writeset);
    if writetx(&readset, &writeset, txp) >= 0
        break;
    if errno != ETX
        exit;
    /* そうでなければリセットし再試行する。*/
    /* いずれか他のプロセスがread()以降にf1またはf2を変更した。*/
    resettx(&fdset);
```

*アクセスについての上記の例を想起すると、TXを使用する擬似コードは、ファイルf内の各レコードrに対して以下ようになる。

※f3に関する第2の例に対してTXを使用する擬似コードは、readtx()またはwritetx()の失敗後に再試行するときのみ、locSeqNf1もしくはlocSeqNf2またはその両方をリセットする。

【0136】

【0137】プログラムが後でファイルに対してreset x()を呼び出さずにf 1、f 2またはf 3にアクセスする場合、他のプロセスがこの間にそのファイルを変更していれば、アクセスは失敗する。

【0138】〔応用〕TXは、データベースシステムで用いられる楽観的な、タイムスタンプに基づくアルゴリズムに類似した一種のトランザクションの同期をサポートする。楽観的方式では、トランザクションはデータを読み出してタイムスタンプを記録し、計算を行い、データベース更新のリストを作成する。コミット点において、トランザクションは、その更新をタイムスタンプとともにデータベースシステムへ送る。更新とタイムスタンプの組合せがトランザクション同期の孤立性を満足する場合、データベースはそのトランザクションをコミットする。そうでない場合、トランザクションは中断する（そして再始動される）。

【0139】これに対して、悲観的な、ロックに基づく方式では、トランザクションTは各読み出しまたは書き込みの前に、オブジェクトに対する許可（すなわち、ロック）を取得する。ロックは、他のトランザクションによるそのオブジェクトにおける変更を排除する。そのロック要求が他のトランザクションのロックと衝突した場合、Tはそれ以上の計算を行わず、単に、他のトランザクションがそのロックを解放するまで待機する。悲観的アプローチは、中断および再始動によって作業が失われることが少ないため、トランザクションがデータベースオブジェクトに対して集中的な長期間の計算を実行するようなアプリケーションに対しては良好である。

【0140】楽観的方式は、オペレーティングシステムの領域では重要な効果を有する。オペレーティングシステムは通常いくつかのアプリケーションに同時にサービスするため、オペレーティングシステムによって制御される資源への公平なアクセスを各アプリケーションに提供することに注意しなければならない。楽観的方式はロックを待たないため、アプリケーションが資源へのアクセスを拒否される可能性は小さい。

【0141】〔結論〕以上では、どのようにして、ユーザレベルのバックアップファイルシステムを改良して、複製ファイルを有する分散システムを生成するかについて説明した。複製ファイルのローカルコピーに対して実行される順次読み出し操作および書き込み操作は、ただ1つのコピーが存在するファイルに対する読み出し操作および書き込み操作と同じ意味規則を有する。この意味規則は、分散システムの要素上のトークンサーバによって管理される読み出しトークンおよび書き込みトークンを使用する分散同期システムによって実現される。好ましい実施の形態では、分散同期システムは、複製ファイルの各ローカルコピーに対する7個のロックによって実現される。さらにこれらのロックは、3バイトのベクト

ルに対する標準的なオペレーティングシステムロックを使用して実装される。

【0142】上記で説明したのは、発明者が現在知っているユーザレベルの複製ファイルシステムを有する分散システムを実現する最適な形態であるが、多くの変形例が可能である。特に、本発明の原理は、上記のユーザレベルバックアップファイルシステムとは関係のないシステムでも使用可能である。例えば、ここで開示した同期技術は、複製ファイルを指定すること、または、ファイルバックアップ操作を実行することのために使用される技術とはほとんど独立であり、実際、複製ファイルに対する操作を同期させること以外の目的で使用することも可能である。さらに、同期は、好ましい実施の形態で利用したロックプロトコル以外の方法でも実現可能である。

【0143】

【発明の効果】以上述べたごとく、本発明によれば、複製ファイルを保守するために必要な操作が、分散システムのユーザレベルで実装可能となる。その結果、本発明は、特殊なハードウェアや特殊なオペレーティングシステムを必要としない。好ましい実施の形態は、ユーザレベルのバックアップファイルシステムの変更として実装される。

【図面の簡単な説明】

【図1】好ましい実施の形態における同期を実現するために使用されるロックの実装の図である。

【図2】ライブラリがユーザプログラムに対するインタフェースを再定義する方法の概観を示す図である。

【図3】動的リンクライブラリがオペレーティングシステムインタフェースを再定義するために使用可能であることを示す図である。

【図4】動的リンクライブラリがユーザレベル名前空間を提供するために使用可能であることを示す図である。

【図5】動的リンクライブラリを使用したユーザレベルバックアップファイルシステムの概略図である。

【図6】動的リンクライブラリ内のルーチンの概略図である。

【図7】ユーザレベルバックアップファイルシステムの好ましい実施の形態の概略図である。

【図8】カーネルサーバ305(a)によって提供される名前空間とユーザレベル名前空間の間の関係を示す図である。

【図9】ユーザレベル名前空間とカーネルサーバ305(b)によって提供される名前空間の間の関係を示す図である。

【図10】フロントエンド複製ツリー505の詳細図である。

【図11】バックエンドマップ517の詳細図である。

【図12】複製ファイルに対する同期システムの一部の

擬似コードを示す図である。

【図13】同期システムのトークン機構のブロック図である。

【図14】好ましい実施の形態におけるwriteシステムコールを置換するコールの擬似コードを示す図である。

【図15】2要素システムを有する分散システムにおいて複製ファイルに対する操作の同期を示す状態図である。

【図16】好ましい環境で使用されるロックのテーブルの図である。

【図17】図16のロックの意味規則のテーブルの図である。

【符号の説明】

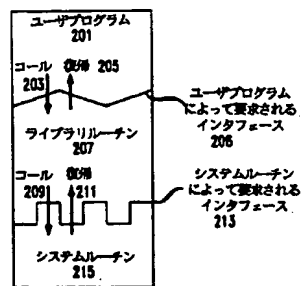
201 ユーザプログラム
203 ファンクション呼出し
205 復帰
206 インタフェース
207 ライブラリルーチン
209 ファンクション呼出し
211 復帰
213 インタフェース
215 システムルーチン
301 システム1
305 カーネルサーバ
306 ユーザプロセス
307 ディスク
309 アプリケーションプログラム (ユーザプログラム)
311 コール
313 復帰
315 オペレーティングシステムライブラリ1
317 コール
319 復帰
321 オペレーティングシステムライブラリ2
323 副次的効果
401 システム
403 オペレーティングシステムライブラリ
405 ユーザレベル名前空間
409 ユーザプロセス
501 ユーザレベルバックアップファイルシステム
503 アプリケーションプロセス
505 フロントエンド複製ツリー (FRT)
507 lib. 3dライブラリ
509 アプリケーションプログラム
511 主システム

512 バックアップメッセージ
513 バックアップシステム
515 バックエンドサーバ
517 バックエンドマップ
601 ルーチン
603 ルーチン名
605 引数
701 ユーザレベルバックアップファイルシステム
703 ログファイル
709 パイプ
710 パイプ
711 パイププロセス
715 システムコールエンジン (SYSCALL ENGINE)
716 バックエンドログプロセス (BLP)
717 モニタ
719 モニタ
801 名前空間
803 サブツリー
901 エントリ
903 ユーザレベル名前空間情報
905 バックアップシステム情報
1003 MAXTRY
1005 INIT
1009 RPLOP配列
1013 RTREE PTR
1015 RTREE
1025 接続サーバ
1027 ファイルディスクリプタキャッシュ
1105 フロントエンドファイル識別子 (FFID)
1106 フロントエンドパス名 (FP)
1107 フロントエンドプレフィクス (FPR)
1108 サブツリーパス名
1111 バックエンドファイルディスクリプタ
1113 パス名マップ
1117 オープン複製ファイルリスト
1118 フロントエンドパス名
1119 バックエンドパス名
1301 分散システム
1302 ビアホスト
1307 トークンファイル
1309 プロセス
1311 トークンサーバ
1325 複製ファイル
1327 書き込みトークン
1328 読み出しトークン

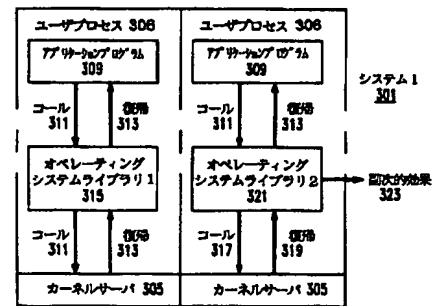
【図1】

システム 1801 ロック	OS ロック	バイト		
		0	1	2
AR	R	-	-	-
SR	R	-	-	-
LW	R	-	-	-
RW	W	-	-	-
WT	W	-	-	-
RT	W	-	-	-
NT	W	-	-	-

【図2】



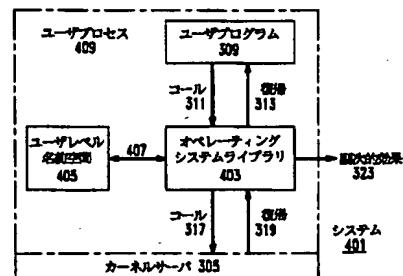
【図3】



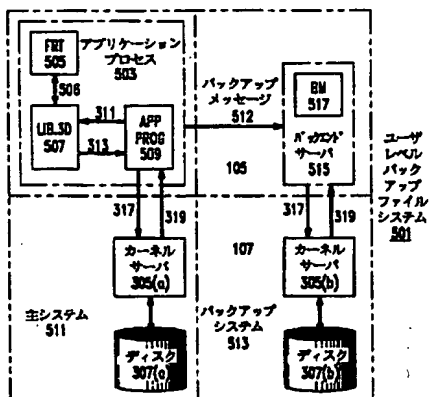
【図17】

ロック	AR	SR	LW	RW	WT	RT	NT
AR			X	X			X
SR	X	X	X	X		X	X
LW	X	X	X	X			
RW			X	X			
WT					X		
RT						X	
NT		X	X				

【図4】



【図5】



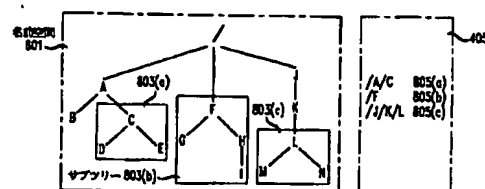
【図6】

```

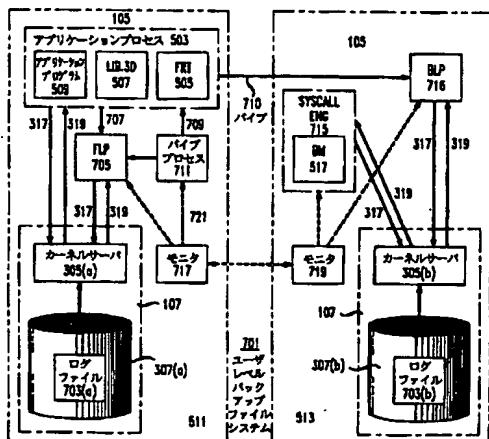
ルーチン名 601
IF (APP-PROG-FILE-OP) (APP-CODE-FILE-OP-ARGS)
  602
  IF (SERVER-FILE-OP) (SERVER-FILE-OP-ARGS)
    603
    IF (IN-REP-FILE) (FILE-ARG)
      604
      SEND-MESSAGE (MESSAGE-ARGS)
      RETURN (SUCCESS)
    ELSE
      RETURN (SUCCESS)
  ELSE
    RETURN (IN-CODE)

```

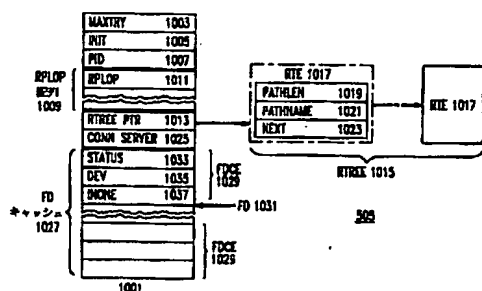
【図8】



【图7】



【図 10】



【图 14】

【图9】



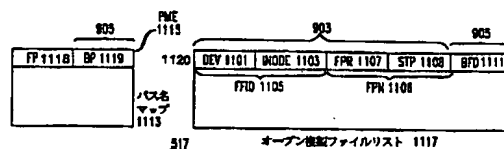
【图 1 2】

```

1  ssize_t write(hd fides, const void *buf, ssize_t nbytes)
2  {
3      get_write_token(<-) 1201
4      r = sync(SFS_WRITE, fides, buf, nbytes);
5      if (r > 0 && !isReplicatedTree(fides))
6          send a message to the remote file system
7      release_token(<-)
8      return (r); 1202
9  }

```

【図 1 1】

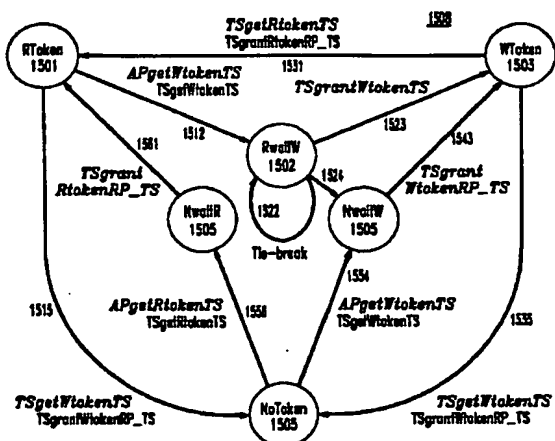


【图 15】

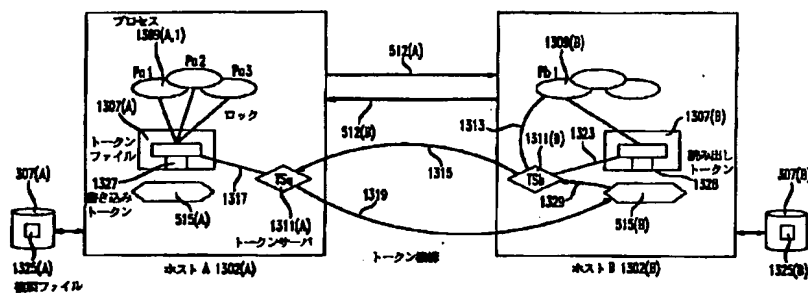
```

1  subr1 write(tst fdlen, const void *buf, strn_t cstyle)
2      {
3      TOKEN_REGION = &tokenstr[fdlen];
4      if (LOCK(TOKEN_REGION_STATE, shared, non-block) == Fail)
5          send a request-to-been message to Token Server
6      Lock(TOKEN_REGION_STATE, shared, block);
7      Lock(TOKEN_REGION_TOKEN, exclusive, block);
8      r = sprintf(STS_write, fdlen, buf, cstyle);
9      if (r > 0 do independent=true(fdlen);
10         send a message to the remote file system
11         Unlock(TOKEN_REGION);
12         return r;
13     }

```



【図13】



【図16】

ロック	要求主体	意味
アトミック読み出し(AR)1601	プロセス 1309	ローカルコピーからのアトミック読み出し
順次読み出し(SR)1603	プロセス 1309	ローカルコピーからの順次読み出し
ローカル書き込み(LW)1605	プロセス 1309	ローカルコピーへの書き込みおよびリモートバックエンドサーバ 515 の待ち行列への送信
リモート書き込み(RW)1607	ローカルバックエンドサーバ 515	ローカルコピーへの書き込み
書き込みトークン(WT)1609	ローカルトークンサーバ 1311	ローカルトークンサーバは複製ファイルに対する書き込みトークンを有する
読み出しトークン(OT)1611	ローカルトークンサーバ 1311	ローカルトークンサーバは複製ファイルに対する読み出しトークンを有する
トークンなし(NT)1613	ローカルトークンサーバ 1311	ローカルトークンサーバは複製ファイルに対するトークンを有しない

フロントページの続き

(72)発明者 アンドレア エイチ. スカーラ
 アメリカ合衆国, 07928 ニュージャージー
 ー, チャタム, オーチャード ロード 26